UNIVERSIDAD
COMPLUTENSE
MADRID

Análisis docente con Machine Learning

Trabajo de fin del grado del Grado en Ingeniería del Software, Facultad de Informática, Universidad Complutense de Madrid, 2018-2019

La presente memoria documenta el proceso de desarrollo de una aplicación de Software para la limpieza y posterior análisis de datos relativos al uso del Campus Virtual Integrado de la Universidad Complutense de Madrid por parte de sus alumnos, mediante algoritmos de minería de datos y aprendizaje automático.

Autor: Morell Prats, Pedro
Profesor director:
Sánchez-Élez Martín, Marcos
Miñana Ropero, Guadalupe
Curso académico: 2018-2019

UNIVERSIDAD COMPLUTENSE MADRID

Facultad de Informática

Índice

Resumen

La presente memoria documenta el proceso de desarrollo de una aplicación de Software para la limpieza y posterior análisis de datos relativos al uso del Campus Virtual Integrado de la Universidad Complutense de Madrid por parte de sus alumnos, mediante algoritmos de minería de datos y aprendizaje automático. Los datos analizados corresponden a ficheros de trazas que contienen información sobre las acciones realizadas en la plataforma, tales como: el usuario que ha realizado la acción, la fecha y hora en la que se ha realizado, y los componentes a los que se han accedido (apuntes, foros, wikis, etc.).

Para ello, se desarrolló un script para el procesamiento y limpieza de datos de entrada, se evaluaron varias librerías de algoritmos de aprendizaje automático, de las cuales se eligió la herramienta Weka, y se desarrolló una aplicación para integrar los dos componentes anteriores, junto con una interfaz de usuario para la ejecución de los algoritmos.

Para la implementación de algoritmos de aprendizaje automático se utilizó la librería Waikato Environment for Knowledge Analysis (WEKA). La aplicación se desarrolló en Java, y la interfaz fue implementada con la biblioteca gráfica Swing. El script de limpieza de datos se desarrolló en Python.

La aplicación desarrollada como parte de este trabajo permite al usuario procesar y analizar los datos de entrada mediante su interfaz gráfica. El usuario puede elegir el fichero a analizar, el algoritmo que se va a aplicar sobre este fichero, y los distintos parámetros para filtrar los datos de entrada. Como resultado, la aplicación genera y muestra por pantalla una gráfica correspondiente a los datos analizados y filtrados.

Palabras clave:

Machine Learning, aprendizaje automático, minería de datos, Weka, ficheros de logs, interfaz de usuario, clustering, Campus Virtual

Abstract

This text documents the process of development of a Software Application for the processing and further analysis of data relative to the use of the integrated online platform "Campus Virtual" of Universidad Complutense de Madrid by its students, via the application of data mining and machine learning algorithms. The data analyzed represents log files that contain information about the actions performed in this platform, such as: the user that performed it, the date and time and the components that were accessed (notes, forums, wikis, etc.).

In order to do this, a script was made for the processing and cleansing of input data, several machine learning libraries were evaluated, of which the one chosen was the Weka suite, and an application was built to integrate these two components, along with a user interface to run these algorithms.

For the implementation of machine learning algorithm, the used library was Waikato's Weka (Waikato Environment for Knowledge Analysis). The application was written in Java, and its interface was implemented using Swing, a widget toolkit for Java. The script for data cleansing was written in Python.

The developed application as part of this project allows the user process and analyze the input data via its graphic interface. The user can choose the file to analyze, the algorithm that's going to be applied, and a series of parameters to filter the input data. As a result, the application generates and displays a graph with the analyzed and filtered data.

Key words:

Machine Learning, data mining, Weka, log files, user interface, clustering, diagrams, Campus Virtual

1. Introducción

La presente memoria describe el trabajo de fin de grado 'Análisis docente con Machine Learning', que consiste en el desarrollo de una aplicación para realizar análisis de ficheros docentes mediante algoritmos de aprendizaje automático.

Este trabajo es la primera parte de un proyecto más amplio originalmente planteado para un equipo de desarrollo más grande. Esta parte se centra en el preprocesamiento de datos, así como el análisis de esos datos mediante técnicas de minería de datos, y serviría como base para un proyecto en el que se integrarían técnicas de predictibilidad y estadística, para poder realizar análisis más profundos y fiables. Por tanto, corresponde necesariamente a la primera fase del proyecto.

1.1. Objetivos

El objetivo de este trabajo es la creación de un sistema que realice varias tareas relacionadas con el procesamiento y análisis de ficheros de logs del Campus Virtual de la UCM. En concreto, la aplicación se compondría de un programa encargado de la limpieza previa de los datos, los algoritmos de análisis implementados y una interfaz de usuario que permita seleccionar los ficheros a analizar, los algoritmos a aplicar sobre estos datos, así como tomar decisiones sobre el preprocesamiento de datos. Como resultado, la aplicación mostraría un diagrama generado a partir de la información introducida por el usuario. Los requisitos del sistema que se quiere construir están en la figura 1.1.

Requisitos del sistema.
1. Proceso automático de limpieza de datos de entrada.
2. Medio visual por el cual el usuario pueda realizar tareas.
3. Algoritmos de aprendizaje automático para el análisis de datos.
4. La visualización de los diagramas generados.

Figura 1.1. Requisitos del sistema.

Para solventar estos requisitos, se han realizado las siguientes tareas:

- El desarrollo de un script en Python que obtenga como entrada un fichero de Excel con las trazas correspondientes a un curso entero del Campus Virtual, y genere un nuevo fichero con la información preparada para su posterior análisis.
- El desarrollo de una interfaz que permita seleccionar qué fichero se desea analizar, que algoritmo se va a aplicar, y, opcionalmente, seleccionar una condición para reducir la cantidad de información procesada (por ejemplo, procesar logs correspondientes solo a unas fechas concretas).
- La búsqueda de una librería que contenga los algoritmos de aprendizaje automático no supervisado necesarios para el análisis de los datos.
- El desarrollo de una aplicación que integre los componentes descritos anteriormente: el script, la interfaz y la librería de algoritmos.
- La gestión de control de versiones del proyecto.

El objetivo final de la aplicación es permitir al profesor o administrador de un curso visualizar grandes cantidades de información de forma rápida, en un solo diagrama, mediante una interfaz sencilla e intuitiva.

1.2. Conceptos generales sobre aprendizaje automático

El aprendizaje automático, también conocido como Machine Learning, es un conjunto de técnicas que proporcionan a un sistema informático la capacidad de aprender y mejorar en base a su propia experiencia, y no por intervención explícita de una persona.

El aprendizaje automático consiste mayormente en el uso de un conjunto de datos de entrenamiento para aprender, y utilizar la experiencia obtenida para análisis y predicitibilad con nuevos conjuntos de datos. Principalmente se utilizan dos métodos, según la naturaleza de los datos de entrada:

- Algoritmos de aprendizaje automático supervisados: se utiliza sobre conjuntos de datos clasificados. A partir del análisis del conjunto de datos de entrenamiento, los algoritmos son capaces de clasificar nuevos datos. Algunos ejemplos de algoritmos supervisados son: regresión logística, árboles de decisión y Support Vector Machines (SVM).

- Algoritmos de aprendizaje automático no supervisados: el objetivo no es entrenar al algoritmo para ser capaz de clasificar nuevos datos, si no analizar los datos de entrada para encontrar estructuras o patrones escondidos. Los datos de entrada, por tanto, no están clasificados o etiquetados. Algunos de los algoritmos más habituales son: algoritmos de clustering, análisis de componentes principales y descomposición en valores singulares.

Las técnicas aplicadas en este proyecto corresponden a técnicas no supervisadas. Los datos de entrada contienen información acerca del uso del Campus Virtual de la UCM por parte de los alumnos y profesores. Cada instancia corresponde a una acción dentro de la plataforma, y contiene una serie de atributos, tales como el usuario que ha realizado la acción, el componente al que ha accedido, la fecha y hora de dicha acción, etc. Estos datos de entrada se explicarán en detalle en secciones posteriores de la memoria.

1.3. Aprendizaje automático en docencia

Las técnicas de aprendizaje automático se utilizan ya en docencia, sobre todo en plataformas de educación online, para analizar y obtener resultados sobre datos como tasas de deserción y retención, rendimiento de estudiantes, etc.[3][4] El uso de estas técnicas ha ido en aumento en los últimos años, y se han aplicado en plataformas como Coursera, FutureLearn o Udacity.[3]

1.4. Aplicación de conceptos obtenidos en la carrera

El desarrollo de este proyecto ha sido posible en gran medida por los conocimientos que se han obtenido gracias a las distintas asignaturas que se han cursado en el Grado de Ingeniería del Software.

Por un lado, se han aplicado conocimientos básicos de aprendizaje automático obtenidos en las asignaturas Aprendizaje Automático y Big Data, e Ingeniería del Conocimiento. Estos han servido para el desarrollo de la aplicación, pero sobre todo han sido de utilidad a la hora de realizar búsquedas de conceptos necesarios para el proceso de documentación.

El diseño de la aplicación se ha realizado teniendo en cuenta conceptos obtenidos a partir de las asignaturas de Ingeniería de Software y Modelado de Software, cursadas en segundo y tercer año respectivamente. Estas han sido de utilidad a la hora de incorporar patrones de arquitectura en la aplicación, y para diseñar los diagramas de clase y secuencia. Para la implementación, se han podido aplicar conocimientos obtenidos en Técnicas de Programación, tales como la programación orientada a objetos y el lenguaje Java.

Para las consideraciones éticas descritas a continuación se ha partido de la base de los conocimientos obtenidos en la asignatura Ética, Legislación y Profesión.

1.5. Consideraciones éticas

Para el futuro uso de la aplicación desarrollada, así como para futuras actualizaciones en la implementación, deben tenerse en cuenta una serie de consideraciones éticas.

En un primer lugar, se debe considerar que se utilizan datos de estudiantes, y aunque en las trazas no se guarda información personal sobre un alumno, y figura un ID en lugar de un nombre, alguien que administre un curso puede averiguar a qué alumno corresponde un ID concreto. Esa visibilidad sobre el comportamiento de un alumno al utilizar la plataforma se debe tener en cuenta como una cuestión de privacidad y confidencialidad.

También debe tenerse en cuenta que una mala implementación de la aplicación puede llevar a que esta genere resultados no fiables, que lleven a un profesor a tomar decisiones en base a conclusiones erróneas.

1. Introduction

This document describes the end-of-degree project "Educational analysis with Machine Learning", which consists in the development of an application that analyzes educational files using machine learning algorithms.

This work is the first part of a bigger project, originally planned for a bigger development team. This part focuses on preprocessing of data, as well as analyzing that data via data mining techniques, and it serves as a starting point for a project in which predictability and statistic techniques would be integrated, in order to perform more profound and reliable analysis. Consequently, it would correspond to the first phase of the project.

1.2. Objectives

The goal of this project is the creation of a system that performs several tasks related to the processing and analysis of log files from the Virtual Campus of the UCM. More specifically, the application would consist of a program that cleanses the input data, the algorithms implemented, a user interface that allows the selection of the file to be analyzed, the algorithm to be applied to this file and other options regarding the processing of the data. As a result, the application would show a graph generated from the information introduced by the user. The system requirements are in figure 1.1.

System requirements.
1. Automatic process of input data cleansing.
2. Visual medium in which a user can perform tasks.
3. Machine learning algorithms for the analysis of data.
4. The visualization of the generated graphs.

Figure 1.1. System requirements.

In order to fulfill these requirements, the next tasks were performed:

- The development of a script in Python that receives an Excel file with the logs of a course in the Virtual Campus as the input and generates a new file with the information prepared for its posterior analysis.
- The development of an interface that allows the selection of the file that's going to be analyzed, the algorithm that's going to be applied, and optionally, a condition to reduce the amount of information processed (for example, process logs corresponding only to certain dates).
- The search for a library that contains the unsupervised machine learning algorithms needed for the analysis of data.
- The development of an application that integrates the components described just now: the script, the interface and the algorithm library.
- The version control for the project.

The final goal for the application is to allow the teacher or course administrator visualize large amounts of information in a quick way, in one single graph, using a simple and intuitive interface.

1.2. General concepts about Machine Learning

Machine learning is a series of techniques that provide a system capacity to learn and improve via its own experience, and not human intervention. It mainly consists in the use of sets of training data for learning, in order to use the obtained experience for the analysis and predictability of new data sets. Two main methods are used, according to the nature of the input data:

- Supervised machine learning algorithms: used upon assorted training data sets. Using the training data set, the algorithm can classify new data. Some examples of supervised algorithms are: logistic regression, decision trees and Support Vector Machines (SVM).

- Unsupervised machine learning algorithms: the goal is not to train the algorithm to classify new data, but to analyze the input data in order to find hidden patterns or structures. The input data is neither assorted or labeled. Some of the more known algorithms are: clustering, principal component analysis and singular value decomposition.

The techniques applied in this project correspond to unsupervised techniques. The input data contains information about the use of the Virtual Campus of the UCM by the students and teachers. Every instance corresponds to an action performed in the platform, and contains a series of attributes, such as the user that performed the action, the component that was accessed, the date and time of said action, etc. This input data will be explained in detailed in later sections of the document.

1.3. Machine learning in education

Machine learning techniques are used already in education, mainly in online education platforms, to analyze and obtain results about desertion and retention statistics, student performance, etc. [3][4] The use of these techniques has grown over the past years, and is applied in platforms such as Coursera, FutureLearn or Udacity. [3]

1.4. Use of concepts learned in class

The development of this project has been possible in great part thanks to the knowledge obtained in the different classes that were taken up in the degree of Software Engineering.

On one side, basic concepts of machine learning obtained in “Machine Learning and Big Data” and “Knowledge Engineering” were used. These were used for the development of the application, but mostly have been useful to perform the research of necessary concepts for the documentation process.

The application has been designed considering concepts obtained from the subjects “Software Engineering” and “Software Modeling”, coursed in second and third year respectively. These have been useful to incorporate architectural patterns in the application, and to design class and sequence diagrams. For the implementation, concepts

obtained in “Programming Techniques” were applied, such as object-oriented programming and the language Java.

For the ethical considerations described hereunder, concepts obtained in the class “Ethics, Legislation and Profession” were used.

1.5. Ethical considerations

For future use of the developed application, as well as updates in the implementation, a series of ethical questions must be considered.

Firstly, it should be considered that students data is being used, and even though logs don’t contain personal information about a student, and the files feature user ids instead of names, someone who administrates a course can find out to whom a certain id belongs. This visibility of the behavior of students must be considered as a question of privacy and confidentiality.

It must also be considered that a wrong implementation of this application could lead to the generation of unreliable results, which could lead a professor to make decisions based on the wrong conclusions.

2. Estudio previo al desarrollo

Antes de comenzar el desarrollo del proyecto ha sido necesario un proceso de estudio para determinar, por un lado, las herramientas que van a utilizarse, y por otro, cómo utilizar estas herramientas para obtener los resultados esperados.

Para el desarrollo del script se ha utilizado Python porque así estaba determinado en los requisitos del proyecto. Por el desconocimiento de este lenguaje se ha tenido que buscar información sobre cómo realizar las siguientes tareas:

- Abrir un documento Excel existente y extraer sus datos.
- Manejar distintos tipos de datos, como fechas, horas y enteros.
- Iterar un conjunto de datos.
- Crear un fichero nuevo csv y escribir sobre él.

Para el desarrollo de la aplicación se ha optado por utilizar Java, al tratarse de un lenguaje que permite de forma sencilla integrar los componentes descritos en el apartado anterior, además de ser el lenguaje en el que está escrita la librería de aprendizaje automático, cuyo proceso de selección se describe a continuación.

A la hora de implementar las técnicas de aprendizaje automático, se ha decidido utilizar una librería que tuviera ya los algoritmos necesarios para el procesamiento de datos, ya que implementar los algoritmos desde cero supone una complejidad y una carga de trabajo añadidas muy difíciles de asumir para este proyecto.

Se han tenido en cuenta principalmente tres herramientas: Cloud AI, la herramienta de Google, TensorFlow, que es la herramienta más extendida de Machine Learning no supervisado, y Weka, otra herramienta de aprendizaje automático escrita en Java.

- Cloud AI[5] se trata de un servicio de Google Cloud que permite la aplicación de machine learning a modelos ya preparados, y la posibilidad de generar modelos personalizados.
- TensorFlow[6] es una biblioteca de software libre desarrollada por Google que permite la realización de varias tareas de aprendizaje automático. Está escrita en Python y C++.

- Weka[8] es una herramienta desarrollada por la universidad de Waikato en Java que contiene varios algoritmos de aprendizaje automático, así como funcionalidades para el preprocesamiento de datos.

La plataforma de Google, Cloud AI se ha descartado al tratarse de una herramienta de pago. La herramienta TensorFlow fue descartada por su complejidad a la hora de implementar los algoritmos necesarios y por su dificultad de integrar en una aplicación de Java (está escrita en Python, y aunque cuenta con una API para su integración en Java, ésta es inestable según la documentación de TensorFlow[8]).

La herramienta Weka está escrita en Java, y proporciona tanto una librería con clases para integración en una aplicación Java, como una interfaz de usuario para ejecutar sus algoritmos. Tras haberse instalado y probado, se decidió utilizar esta herramienta.

En la figura 4.1 se muestra una tabla comparando las características de cada uno de los sistemas considerados.

	Cloud AI	TensorFlow	Weka
Software libre	NO	SI	SI
Fácil integración en Java	SI	NO	SI
Interfaz de usuario	SI	NO	SI

Figura 4.1. Tabla de características de herramientas de Machine Learning.

Una vez instalada Weka, se han tenido que probar sus distintas funcionalidades para decidir cuáles de ellas pueden implementarse en la aplicación, o pueden resultar más interesantes para el proyecto. En la sección 6, “Weka”, se explica cuales se han implementado y por qué.

En la siguiente sección se explica de forma detallada el desarrollo del script de procesamiento de datos de entrada utilizando Python.

3. Procesamiento de datos de entrada

El fichero que se utiliza como datos de entrada describe las acciones realizadas por los usuarios de un curso en el campus virtual de la UCM durante un año, tales como las páginas, secciones o foros que han visitado los alumnos, así como las secciones creadas o actualizadas por parte de los administradores. Este es un fichero de tipo xls que contiene como campos los siguientes: fecha, hora, contexto del evento (que toma valores como "Curso: FC Grupo E", "Archivo: Nota Final Primer Parcial", o "Wiki: Apuntes de clase", por ejemplo), componente (por ejemplo "Sistema", "Wiki" o "Carpeta"), nombre del evento ("Curso visto", "Perfil de usuario visto", etc.), y descripción. De este último campo se obtienen tres nuevos campos, id de usuario, descripción y módulo, mediante un proceso descrito en la siguiente sección, con el objetivo de aportar información más detallada en los resultados obtenidos por la aplicación.

Para poder realizar el análisis de los datos de entrada es necesario transformar estos datos a un formato que pueda ser utilizado por Weka. Esta limpieza es un proceso automático que se realiza mediante un programa desarrollado en Python, que obtiene el fichero (.ods) inicial, lo recorre por filas y lo amplía con nueva información extraída a partir del campo "descripción".

Esta nueva información se trata de tres campos: código de usuario, acción realizada y código del módulo accedido por el usuario (será NULL en los casos en los que no se haya accedido a un módulo). A continuación, se muestran cuatro ejemplos, donde el campo original de la descripción está en negrita y los campos nuevos obtenidos están en la siguiente línea:

- **The user with id '68626' viewed the 'wiki' activity with course module id '2669837'.**
  68626 | viewed the 'wiki' activity with course | 2669837
- **The user with id '395023' has viewed the discussion with id '517005' in the forum with course module id '2440070'.**
  395023 | has viewed the discussion with id '517005' in the forum with | 2440070

- **The user with id '391864' viewed the section number '3' of the course with id '76684'**.
  391864| viewed the section number '3' of the course | NULL
- **The user with id '391864' viewed the 'folder' activity with course module id '2741353'.**
  391864 | viewed the 'folder' activity with course | 2741353

Estos nuevos campos son insertados, junto con los campos originales, en un fichero csv, ya que este es uno de los formatos aceptados por Weka, para su posterior análisis mediante algoritmos de minería de datos. En la figura 3.1 se muestra un ejemplo claro de esta transformación.

Fecha y hora	Contexto	Componente	Nombre	Descripción
26/07/2017 10:46	Curso: FC Grupo E	Registros	Informe de registros visto	The user with id '68626' viewed the log report for the course with id '76684'.

Fecha	Hora	Contexto	Componente	Nombre	User id	Descripción	Module id
26/07/2017	10:46	Curso: FC Grupo E	Registros	Informe de registros visto	68626	viewed the log report for the course	NULL

Figura 3.1. Transformación de datos de entrada.

En términos generales, el algoritmo que implementa este script es un bucle que itera el fichero por filas y realiza distintas operaciones. En un primer lugar comprueba que la fila cumple con los requisitos necesarios para ser procesada. Estos requisitos corresponden a una condición introducida por el usuario para filtrar los datos. A continuación, se extraen

los campos del fichero, se separa el campo fecha en fecha y hora, y se procesa el campo descripción para generar los siguientes campos:

Id del usuario

El id del usuario es siempre la cuarta palabra de la descripción, por tanto, tras haber transformado la descripción en un array con una palabra por posición, basta con acceder a la cuarta posición para obtener este campo.

Descripción

El campo descripción se recorre a partir del id del usuario y hasta el final o hasta que el programa encuentre la palabra 'course'. Si la descripción contiene la palabra 'course' hay dos opciones, o bien especifica el id del curso, el cual es irrelevante ya que el fichero corresponde a un solo curso, o bien especifica el id del módulo. En los ejemplos anteriores se muestra un caso de cada.

Id del módulo

Si la descripción contiene la palabra 'module', el id del módulo se encuentra en la última posición del array.

Si el usuario no ha accedido a ningún módulo, este campo contendrá la palabra "NULL".

Tras haber obtenido los campos nuevos, se genera la fila con todos los campos y se escribe al final del fichero csv.

Filtros

Como se ha explicado anteriormente, otra de las funcionalidades de este script es filtrar la información del fichero de entrada. El objetivo de este filtro es permitir al profesor o administrador de un curso analizar solamente una fracción de los datos. Por ejemplo, el profesor puede querer visualizar información relevante al primer semestre de un curso, o a una franja horaria concreta.

De la forma que está implementada el script, el usuario puede filtrar la información por 'hora' y por 'fecha' (con la posibilidad de implementar en el futuro nuevos filtros),

teniendo ambos dos parámetros, fecha/hora inicial, y fecha/hora final. Tanto el tipo de filtro como el parámetro inicial y final vienen dados en el script como argumentos. Por ejemplo, si el administrador desea filtrar por fecha, y quiere visualizar información correspondiente al periodo de exámenes del primer cuatrimestre, los argumentos quedarían así:

- Argumento 1: “fecha”
- Argumento 2: “08/01/2019”
- Argumento 3: “28/01/2019”

Otro ejemplo, mediante el cual se visualizarían resultados basados en trazas registradas entre las 10:00 y las 11:00 a lo largo del curso, sería el siguiente:

- Argumento 1: “hora”
- Argumento 2: “10:00”
- Argumento 3: “11:00”

Tras haber procesado el campo ‘descripción’, el bucle inserta la nueva fila, con los nuevos campos obtenidos en el nuevo fichero.

Ejecutar este script sobre el fichero de muestra da como resultado un fichero csv con el formato que se muestra en la figura 3.2.

```
"fecha","hora","contexto","componente","nombre","user id","descripcion","module id"
"26/07/2017","0:46","Curso: FC Grupo E","Registros","Informe de registros visto","68626","viewed the log report for the course","NULL"
"26/07/2017","10:45","Curso: FC Grupo E","Sistema","Actividad reciente vista","68626","viewed the recent activity report in the course","NULL"
"26/07/2017","10:44","Curso: FC Grupo E","Sistema","Actividad reciente vista","68626","viewed the recent activity report in the course","NULL"
"26/07/2017","10:44","Curso: FC Grupo E","Sistema","Curso visto","68626","viewed the section number '-1' of the course","NULL"
```

Figura 3.2. Extracto del csv generado.

4. Weka

Para realizar el análisis de los datos, una vez procesados por el script descrito en la sección anterior, ha sido necesario encontrar una librería capaz de aplicar varios algoritmos de aprendizaje automático y de generar los diagramas y componentes visuales necesarios para el proyecto. Para ello se ha decidido utilizar la librería Weka (Waikato Environment for Knowledge Analysis).

Weka es una herramienta de aprendizaje automático y minería de datos, programada en Java y desarrollada en la Universidad de Waikato, en Nueva Zelanda. Puede utilizarse mediante su interfaz de usuario, o como librería en un proyecto de Java. La interfaz principal, y de la que se hablará en esta sección es Explorer[9], pero Weka también dispone de tres interfaces adicionales: Simple CLI, Experimenter y Knowledge Flow.[9]

La interfaz Explorer cuenta con seis pestañas, cada una de las cuales contiene una funcionalidad.[8] Éstas son:

- 'Preprocess': permite importar datos de una base de datos o ficheros, y pre-procesar estos datos con algoritmos de filtrado.
- 'Classify': permite aplicar algoritmos de clasificación estadística.
- 'Cluster': dispone de varios algoritmos de clustering, como el de K-means, utilizado en el proyecto.
- 'Associate': da acceso a reglas de asociación para identificar las interrelaciones entre los atributos de los datos.
- 'Select attributes': contiene algoritmos para identificar los atributos más predictivos en un conjunto de datos.
- 'Visualize': muestra un diagrama con puntos dispersos que permiten ser clicados para mostrar información detallada de cada instancia.

En el proyecto se han implementado dos de ellas, 'Cluster' y 'Visualize', las cuales se han probado previamente mediante la interfaz. Aunque en un futuro podrían implementarse nuevas funcionalidades, en este trabajo se han desarrollado sólo estas dos. A continuación, se describe su funcionalidad en detalle.

- Visualize: Esta función proporciona una visión plana de los datos de entrada, sin ningún tipo de clasificaciones o agrupamiento. El diagrama generado muestra la información del fichero tal cual es, pero en una sola imagen en lugar de en un documento de cientos (o miles) de líneas. Era esencial, por tanto, que estuviese implementado en la versión inicial de la herramienta.
- Cluster: Esta permite la agrupación de instancias que tienen características similares. Tiene sentido su implementación simplemente por lo extendido que está su uso en el mundo del aprendizaje automático. Se puede implementar mediante varios algoritmos. El que se ha utilizado en la aplicación es el de k-medias.

En las figuras 4.1 y 4.2 pueden observarse capturas de ambas funcionalidades. Su implementación en la aplicación se detallará más adelante, en la Sección 9, 'Implementación'.

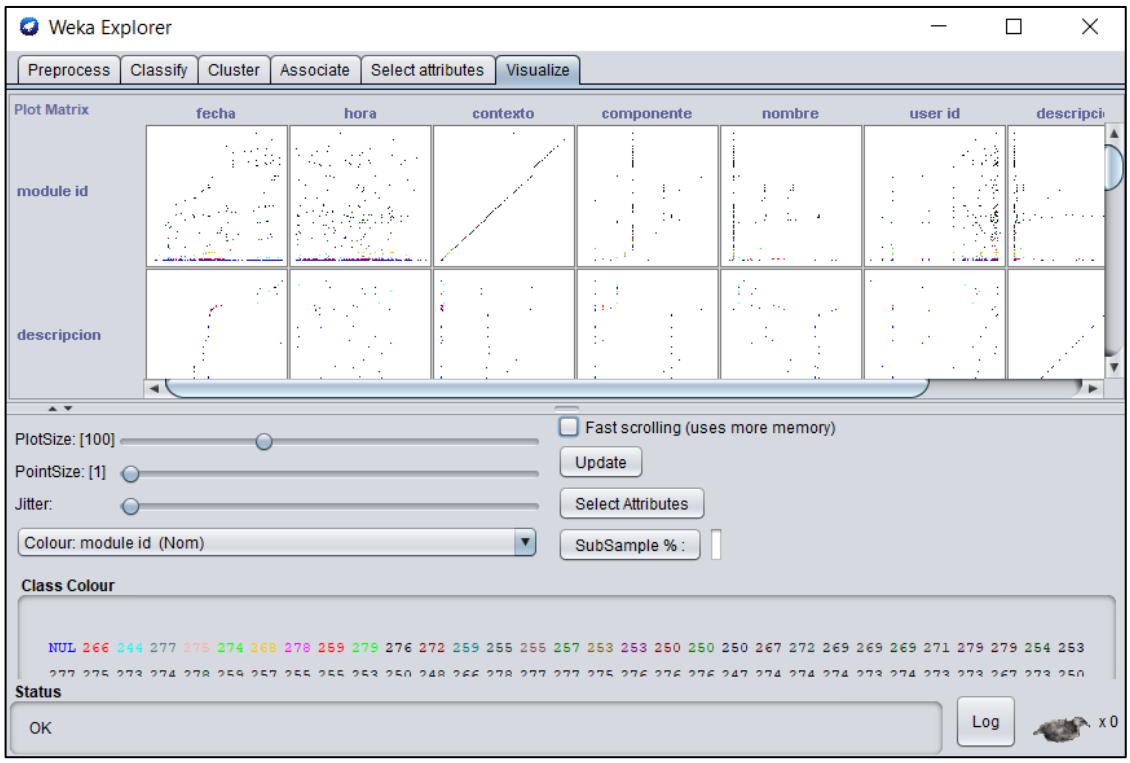


Figura 4.1. Interfaz Explorer de Weka: Visualize.

La interfaz en la figura 4.1 permite seleccionar uno de los diagramas generados, según el atributo que se utilice para el eje *x* y el eje *y*. Además, permite cambiar parámetros como el tamaño del diagrama, de los puntos, el *Jitter* (permite ver con más claridad cuando hay varios puntos situados muy cerca), y el parámetro *Color*, que indica qué atributo dicta el color de los puntos en la gráfica.

La interfaz de la figura 4.2 muestra la funcionalidad *Cluster* de la herramienta, que permite seleccionar el algoritmo, el modo de clustering, y los atributos que deben ignorarse, y visualizar los resultados.

En esta sección no aparecen capturas de los diagramas obtenidos con ambos algoritmos porque estos aparecerán más adelante, en la Sección 11, 'Resultados obtenidos'. Se debe recalcar también, que estas interfaces no se han utilizado en la implementación final de la aplicación, pero han sido necesarias y útiles para determinar qué funcionalidades implementar en la aplicación, y comprobar su viabilidad y eficiencia.

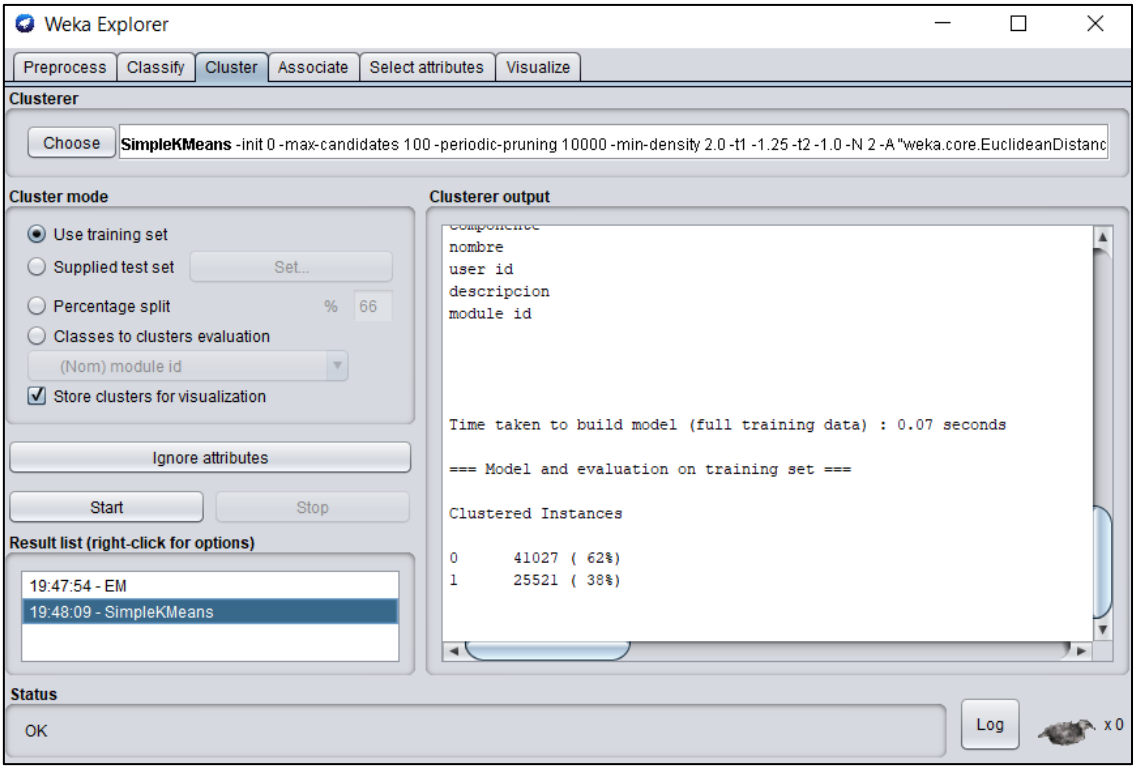
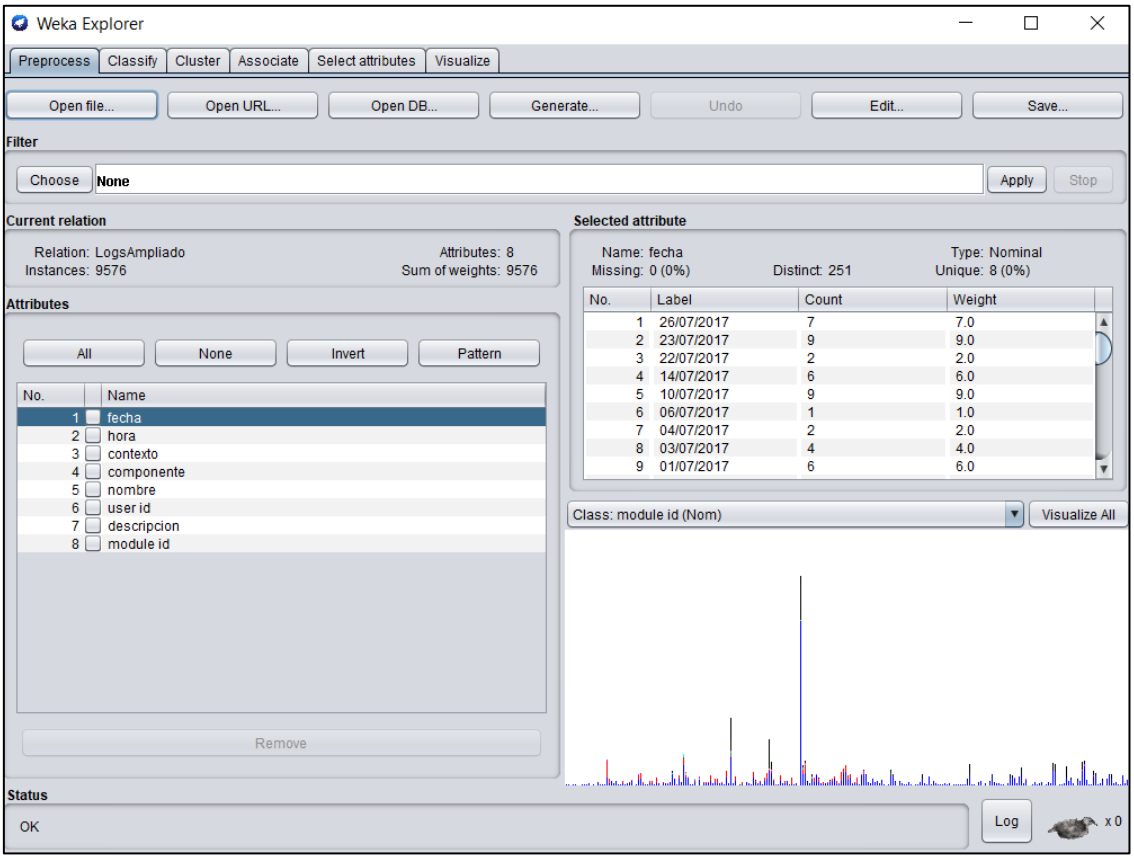


Figura 4.2. Interfaz Explorer de Weka: Cluster.

De la interfaz de usuario también se ha utilizado la funcionalidad Preprocess, para poder cargar el fichero csv, y comprobar que los atributos y los valores del fichero se cargan correctamente. En la figura 4.3. se muestra una captura de esta interfaz con los datos cargados. Los datos de entrada son los obtenidos en el fichero csv que ha generado el script descrito en el apartado anterior.



4.3. Interfaz Explorer de Weka: Preprocess.

4.1. Clases utilizadas

A la hora de integrar las funcionalidades anteriores en la aplicación, no hay una implementación directa y sencilla tal y como se realiza en la interfaz. El objetivo de la implementación es replicar las funcionalidades mediante los métodos que se encuentran en las muchas clases de la librería para Java. A la hora de determinar que clases eran necesarias para cada función, se ha consultado la documentación de Weka[8]

A continuación, se listan y explican las clases Java de la librería Weka que se han utilizado en el proyecto:

CSVLoader

Su función es leer un fichero csv, es decir de valores separados por coma.

Instances

Esta clase se encarga de gestionar un conjunto ordenado de instancia (siendo una instancia cada una de las filas del fichero).

SortLabels

Un filtro sencillo para ordenar las etiquetas de los atributos nominales.

PlotData2D

Sirve para generar el diagrama en dos dimensiones a partir de un objeto de la clase Instances.

VisualizePanel

Un componente visual que hereda de la clase JPanel de Swing (Java). Permite al usuario visualizar un conjunto de datos en dos dimensiones.

JPEGWriter

Toma cualquier objeto de tipo JComponent (Swing) y genera como salida un fichero jpg, mediante una captura de pantalla.

SimpleKMeans

Implementa el algoritmo de agrupación (clustering) k-medias.

ClusterEvaluation

Clase para evaluar modelos de agrupación (generados por la clase anterior).

ClustererAssignmentsPlotInstances

Genera un diagrama en dos dimensiones a partir del modelo generado y evaluado por las dos clases anteriores.

La forma en la que se han utilizado estas clases para la implementación de algoritmos en la aplicación se explica en la Sección 7, “Implementación”.

5. Diseño de la aplicación

Para el diseño de la aplicación, se ha buscado un modelo que permita integrar los componentes descritos anteriormente–el script de procesamiento de datos y los algoritmos implementados utilizando las clases de Weka–en un sistema con una interfaz de usuario donde se pueda seleccionar el fichero a procesar, el algoritmo a aplicar sobre estos datos y los parámetros que se visualizarán en el diagrama.

Los requisitos del diseño, por tanto, serían los siguientes:

- Una clase que implemente una interfaz de usuario con los inputs necesarios.
- Un controlador que envíe una petición del usuario a la lógica de negocio.
- Una clase que contenga la lógica que llama al script desarrollado en Python de limpieza de datos.
- Una clase que contenga toda la lógica algorítmica de aprendizaje automático.

A continuación, se muestra el diagrama de clases de la aplicación, así como los diagramas de secuencia a alto nivel correspondientes a las funciones principales implementadas. Estos diagramas han sido diseñados utilizando Modelio.

5.1. Diagrama de clases

El diagrama de la figura 5.1 muestra todas las clases implementadas por el autor. Las clases de librerías externas al proyecto no se muestran. El razonamiento detrás de este diseño es el de centralizar la lógica relativa a cada aspecto de la aplicación en una clase, así como modularizar y desacoplar la implementación. Por este motivo, toda la lógica de procesamiento de datos de entrada se encuentra en la clase LogsCleaner, y toda la lógica relativa al uso de la librería Weka se ha agrupado en la clase WekaDriver. La clase ResultAL implementa la función que se ejecuta al pulsar el botón “Obtener resultados” de la interfaz, la cual valida que la petición sea correcta y la envía al controlador.

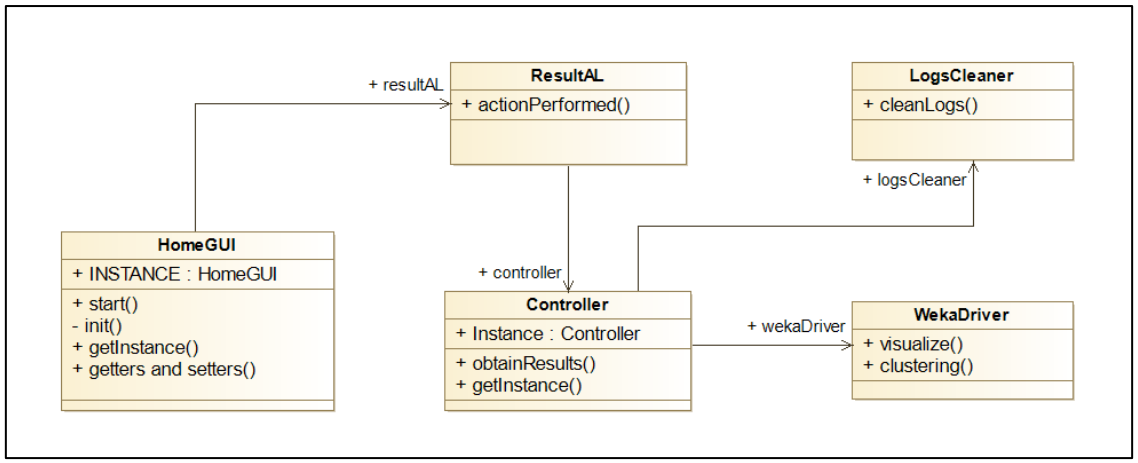


Figura 5.1. Diagrama de clases de la aplicación.

5.2. Diagramas de secuencia

A continuación, se muestran una serie de imágenes que corresponden a diagramas de secuencia que especifican el diseño de cada una de las funciones principales de la aplicación. Estas funciones son:

- actionPerformed: contiene la lógica que se ejecuta cuando el usuario hace clic en el botón "Obtener resultados" de la interfaz, tras haber rellenado los campos requeridos. Se encuentra en la clase ResultAL.
- obtainResults: es una función de la clase controlador que recibe como parámetros los datos introducidos por el usuario en la interfaz y se encarga de llamar a las funciones de las clases LogsCleaner y WekaDriver.
- visualize: es una de las funciones en la clase WekaDriver. Implementa una de las funcionalidades de Weka. Cabe recalcar que, aunque utilice clases de Weka, su implementación en si es propia del autor de este proyecto.
- clustering: la otra funcionalidad de Weka implementada.

Las figuras 5.2 describe la función actionPerformed, que recupera la instancia de la vista principal, obtiene los parámetros que ha introducido el usuario, y envía la petición al controlador. En la figura 5.3 se muestra el diseño de obtainResults del controlador, que instancia la clase LogsCleaner para el procesamiento de datos, y posteriormente WekaDriver para la aplicación del algoritmo escogido.

5.2.1. actionPerformed (**ResultAL**)

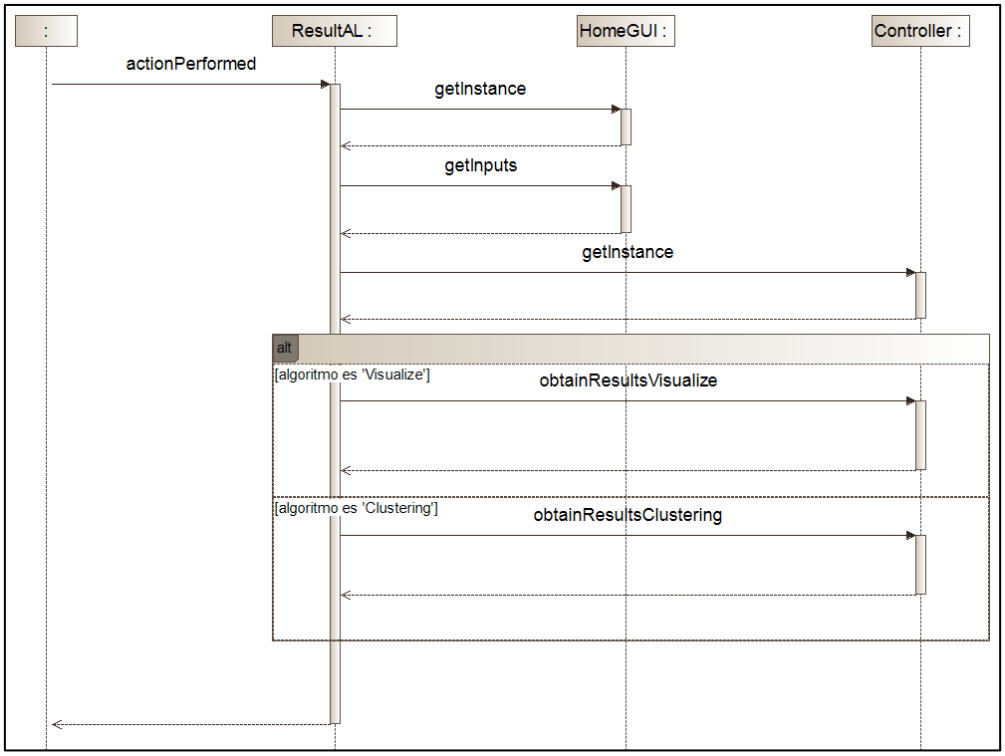


Figura 5.2. Diagrama de secuencia de actionPerformed.

5.2.2. obtainResults (**Controller**)

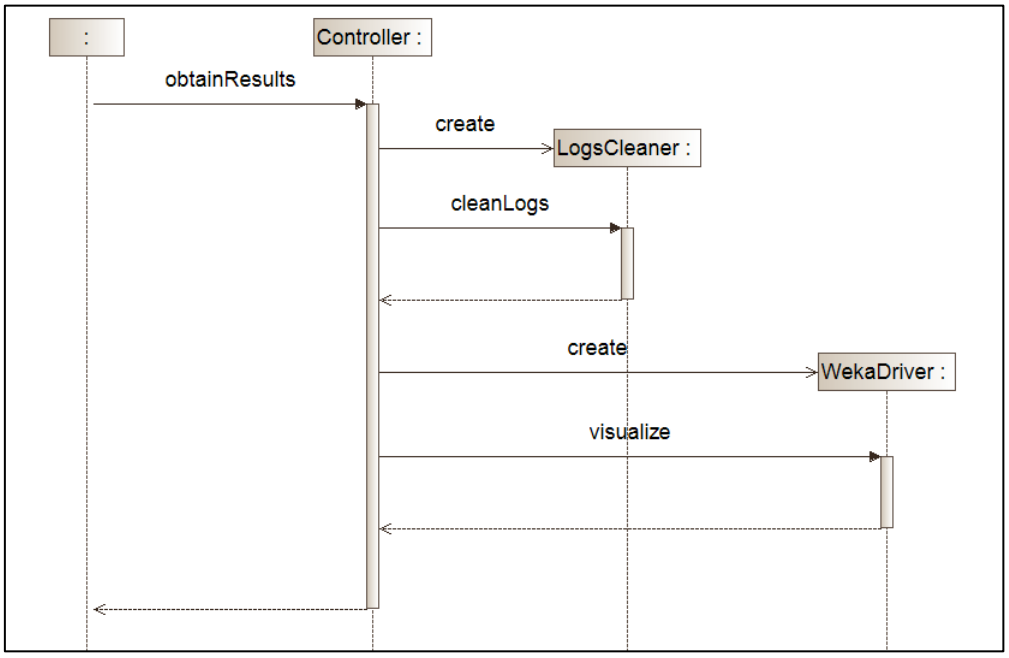


Figura 5.3. Diagrama de secuencia de obtainResults.

5.2.3. visualize (**WekaDriver**)

El diagrama de secuencia de la figura 5.4. especifica la implementación de la funcionalidad "Visualize". El objetivo de esta función es replicar la función "Visualize" de la interfaz de Weka mediante sus clases de Java. Su implementación se describe en detalle en la sección "Implementación".

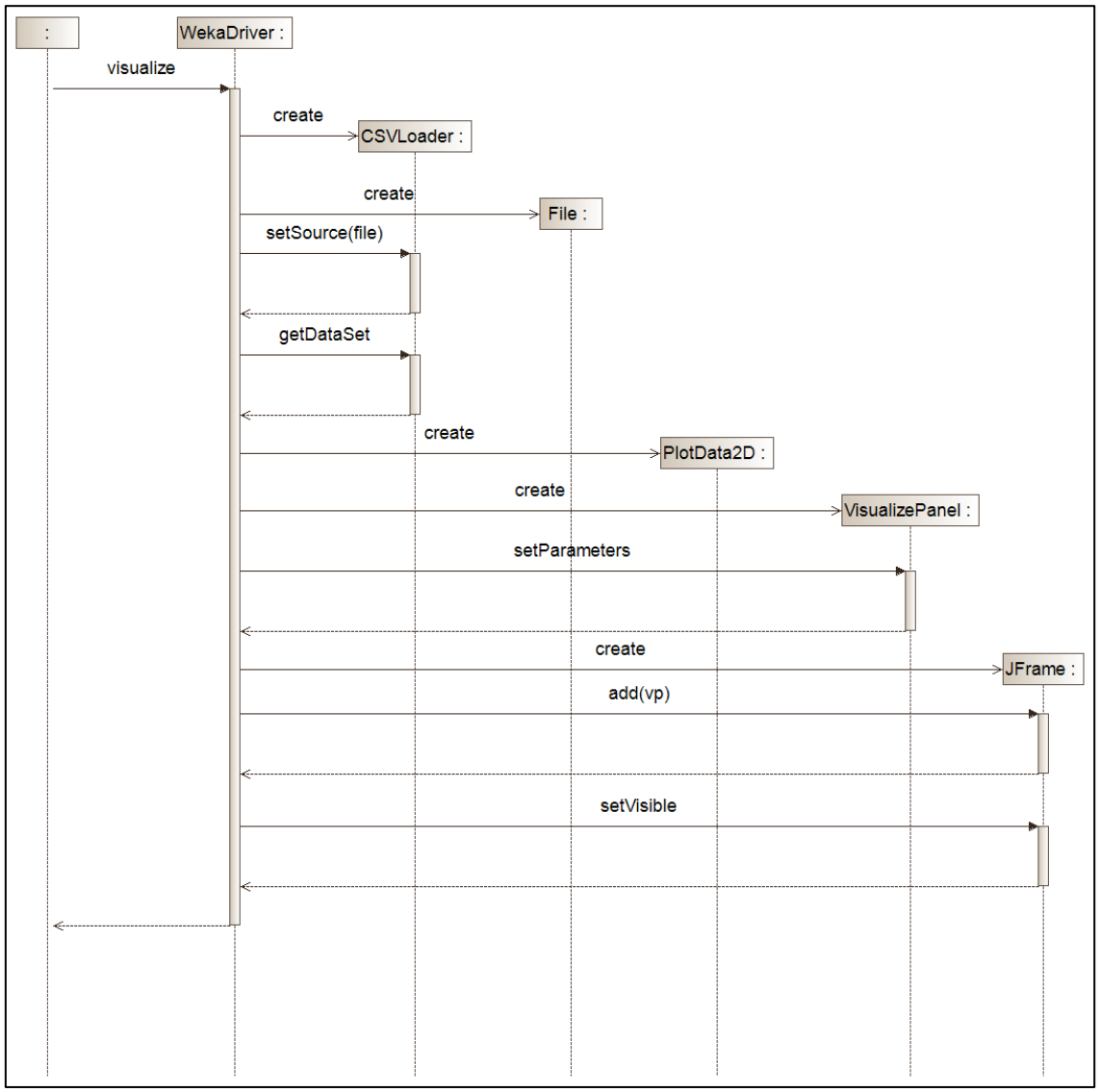


Figura 5.4. Diagrama de secuencia de 'Visualize'.

5.2.4. clustering (**WekaDriver**)

El diagrama de secuencia de la figura 5.5. especifica la implementación de la funcionalidad "Clustering". El objetivo de esta función es replicar la función "Cluster" de la interfaz de Weka mediante sus clases de Java. Su implementación se describe en detalle en la sección "Implementación".

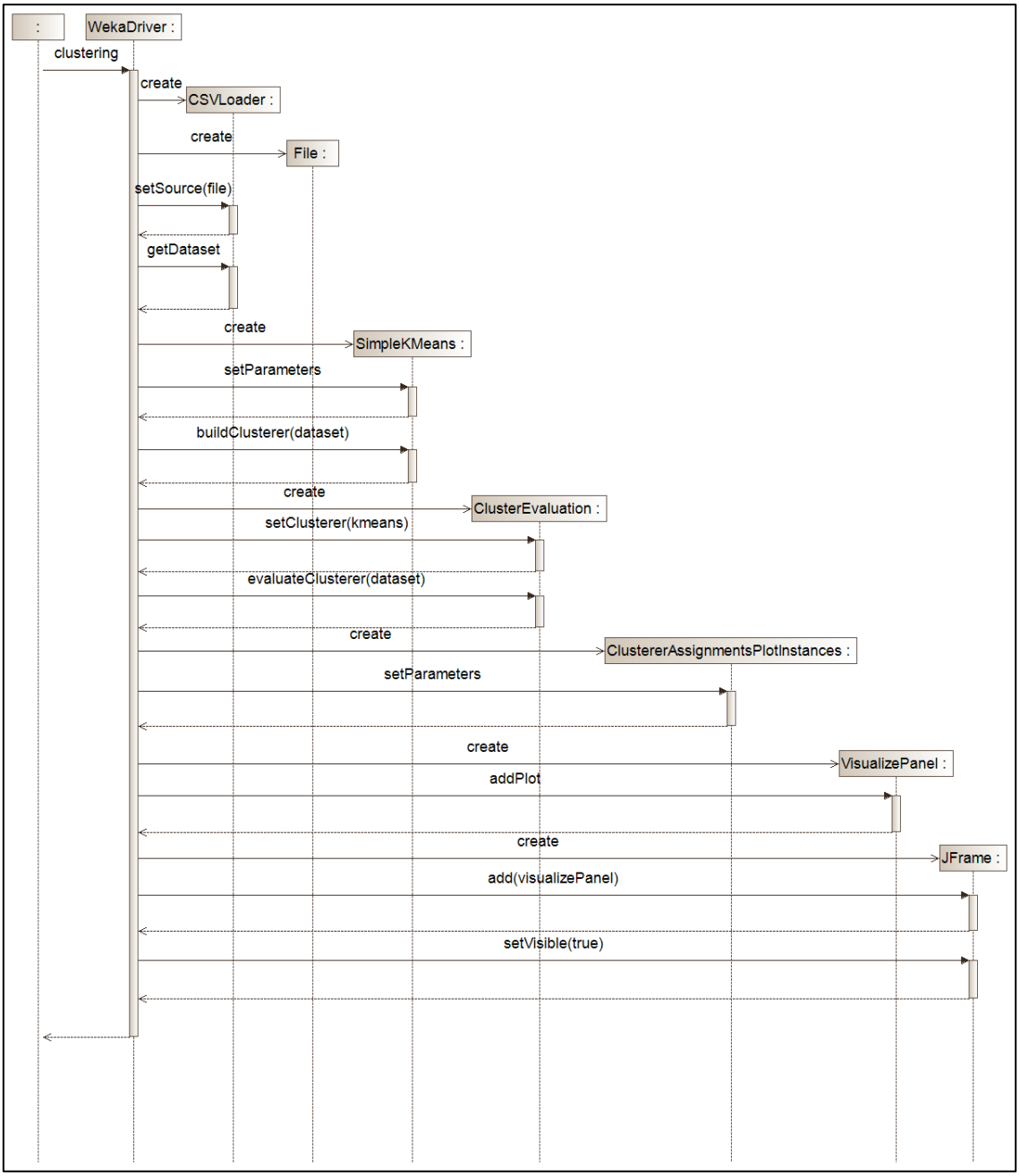


Figura 5.5. Diagrama de secuencia de 'Clustering'

6. Estructura del proyecto

El diseño descrito anteriormente se ha implementado como un proyecto en Eclipse. La estructura del proyecto es la siguiente:

6.1. Estructura del proyecto

Figura 6.1. Estructura de paquetes de la aplicación en Eclipse.

Como se observa en la figura 6.1, el proyecto está compuesto por cuatro paquetes:

- Un paquete con el controlador de la aplicación.
- Un paquete con la clase *Main*, que inicia la aplicación.
- Un paquete con la lógica de negocio.
- Un paquete con las vistas de la aplicación.

Asimismo, el proyecto cuenta con una carpeta, 'python', que contiene los ficheros de datos a procesar y el script de Python, 'limpiar.py', que los procesa y transforma en un csv. Dicho csv tiene como nombre 'LogsAmpliado.csv' y se encuentra también en la carpeta principal.

También se puede observar en el directorio principal la imagen 'logo.png', que corresponde al logo de la Complutense que figura en la interfaz principal, el fichero ejecutable 'CVLogsApp.jar', y la librería referenciada 'Weka.jar'.

6.2. Control de versiones

Durante el desarrollo del proyecto se ha mantenido un repositorio online utilizando el sistema de control de versiones GIT y alojado en la plataforma GitHub. Para realizar operaciones sobre el repositorio se ha utilizado la herramienta SourceTree, como preferencia personal, en lugar de la herramienta para el control de versiones integrada en Eclipse.

Esta gestión de versiones se ha utilizado principalmente como medida de seguridad, para tener en todo momento una copia del proyecto almacenada en la nube. La URL de este repositorio es https://github.com/pedromorell/JavaCVLogs.git.

7. Implementación

La implementación de la aplicación se ha realizado utilizando Java 8, y Eclipse como IDE. A continuación, se explica, de forma general, la implementación de las distintas clases del proyecto.

7.1. Clase Main del proyecto

El método Main del Proyecto recupera la instancia de la vista principal, HomeGUI, y llama a su método *start()*, que se encarga de hacer visible el marco principal de la vista, como se explica en el siguiente apartado.

7.2. Vistas de la aplicación

La aplicación consta de dos vistas, una vista principal que permite al usuario seleccionar los datos que desea analizar, y otra vista con los resultados obtenidos.

La vista principal, denominada como HomeGUI, está compuesta por una cabecera, un input para seleccionar fichero, un input para seleccionar algoritmo, un input para el filtrado de información, inputs adicionales en función del algoritmo escogido ('Visualize' permite seleccionar tres parámetros y 'Clustering' muestra un input para introducir el número de clústeres), y un botón de "Obtener resultados", el cual tiene un ActionListener que se encarga de recuperar la instancia del controlador y llamar al método correspondiente para obtener los resultados. Todos los inputs son de tipo *ComboBox*. En el caso de selección de ficheros, el programa busca los archivos existentes en el directorio 'ficheros' y muestra sus nombres como opciones de selección.

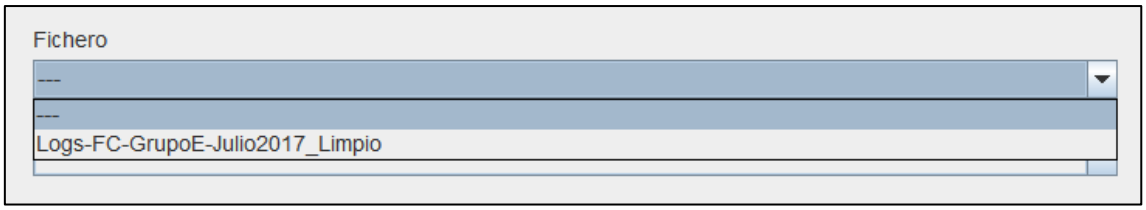


Figura 7.1. Captura del input de ficheros de la interfaz principal.

En la figura 7.1 se muestra el input de ficheros, en el caso de que solo exista el fichero *Logs-FC-GrupoE-Julio2017_Limpio* en el directorio.

La clase tiene un atributo estático *Instance* que permite su implementación como 'Singleton', de forma que solo pueda existir una instancia de esta clase a la vez.

La vista de resultados está compuesta por el diagrama generado y distintos inputs para modificar los parámetros (*x*, *y*, y *color*) mostrados en la gráfica.

7.3. ActionListener

Para la llamada al controlador se ha creado una clase que extiende de ActionListener. Dicha clase se encarga de recuperar los valores insertados por el usuario de la clase HomeGUI, comprobar que estos no son nulos o incorrectos y transformar sus valores en su índice correspondiente en cada fila del csv, ya que este índice será el valor que se utilice por la librería Weka para generar el diagrama.

Tras haber calculado los índices correspondientes se llama al método *obtainResults* del controlador y finalmente se crea la vista con los resultados.

7.4. Controlador de la aplicación

El controlador de la aplicación se ha implementado como *Singleton*, ya que debe existir una sola instancia al mismo tiempo. El controlador tiene dos métodos llamados *obtainResultsVisualize*, y *obtainResultsClustering*, cada uno de los cuales instancian dos clases, una llamada *LogsCleaner* que se encarga de ejecutar el script de python que procesa los datos de entrada, y otra, *WekaDriver*, que genera el diagrama resultado.

7.5. Integración de script de proceso de datos

Para la integración del script de python se utiliza una clase, *LogsCleaner*, que implementa un método, *cleanLogs*.

Este método recibe como argumento el nombre del fichero a procesar, y ejecuta el comando *python limpiar.py nombre-de-fichero.xlsx*. A continuación, el método procede a imprimir por consola el 'output estándar' y el 'error estándar' resultado de ejecutar el comando anterior. El fichero *limpiar.py* contiene el script descrito en la sección 5 de esta memoria.

7.6. Integración de la librería Weka con la aplicación

La integración de la librería Weka, que contiene los algoritmos de aprendizaje automático utilizados en la aplicación, se ha realizado mediante una clase denominada 'WekaDriver'. Esta clase implementa dos funcionalidades distintas, el algoritmo 'Visualize' y el algoritmo 'Clustering'.

Dado que implementar las funcionalidades de Weka en Java ha sido uno de los trabajos de más peso durante el proyecto, a continuación, se muestra el código correspondiente a estas funciones junto con una explicación más detallada de su implementación. Esta implementación es importante también porque muestra las clases pertenecientes a Weka que van a utilizarse, que están indicadas en negrita.

7.6.1. **Visualize**

En la figura 7.2 se muestra la implementación de la primera funcionalidad, denominada 'Visualize', ya que este es el nombre que utiliza Weka. Esta función toma como parámetros tres enteros, cada uno de los cuales corresponde a uno de los atributos, o campos, del fichero csv. El primer parámetro será el atributo que se utilizará en el eje x, el segundo, en el eje y, y el tercero corresponde al color de los puntos mostrados, siendo cada color uno de los valores que puede tomar este atributo. Por ejemplo, si el usuario elige el atributo 'Componente' para color, cada color corresponderá a uno de los valores de este atributo, siendo estos valores 'Sistema', 'Registros', 'Wiki', etc.

El primer paso en esta función es crear un objeto de la clase **CSVLoader**, y un objeto File con el fichero a analizar. Estos dos se utilizan para generar el objeto de tipo **Instances**, que contiene todos los puntos, cada uno de los cuales corresponde a un conjunto de atributos, que van a mostrarse en la gráfica. El componente visual correspondiente a la gráfica es un objeto de la clase **PlotData2D**, el cual se genera con el objeto **Instances**, y se añade posteriormente a un panel de tipo **VisualizePanel**. A continuación, se establecen los parámetros *x*, *y*, y *color* a partir de las variables que se han pasado como argumento, y finalmente se crea un marco (JFrame) donde se añade el panel

con el diagrama y se muestra por pantalla. Adicionalmente se crea una imagen jpg donde se guarda el diagrama generado, a través de la clase **JPEGWriter**.

```
public void visualize(int x, int y, int c) {
	CSVLoader loader = new CSVLoader();
	File file = new File("LogsAmpliado.csv");
	try {
		loader.setSource(file);
		Instances data = loader.getDataSet();
		PlotData2D pd1 = new PlotData2D(data);
		pd1.m_displayAllPoints = true;

		VisualizePanel vp = new VisualizePanel();
		vp.addPlot(pd1);

		vp.setXIndex(x);
		vp.setYIndex(y);
		vp.setColourIndex(c);

		// Abrir diagrama en un Jframe
		Jframe jf = new Jframe("Diagrama");
		jf.setSize(1024, 512);
		jf.add(vp);
		jf.setLocationRelativeTo(null);
		jf.setVisible(true);

		// Guardar diagrama en fichero jpg
		File file1 = new File("src\\imagen.jpg");
		JPEGWriter writer = new JPEGWriter(vp, file1);
		writer.generateOutput();


	} catch (IOException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	} catch (Exception e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
}
```

7.2. Implementación de la función 'Visualize'.

7.6.2. **Clustering**

En la figura 7.3 se muestra el código correspondiente al segundo algoritmo implementado, el de 'Clustering'. La función de este algoritmo es clasificar la información de entrada en distintos clústeres con conjuntos de atributos similares. Cabe recalcar que lo que se está agrupando son instancias. Cada instancia tiene un conjunto de atributos: fecha, usuario, componente, evento, etc. En revisiones futuras de la aplicación podría modificarse esta funcionalidad para poder hacer agrupaciones por usuario, por ejemplo, de manera que cada punto en un clúster sea un usuario distinto, y por tanto cada agrupación corresponda a un conjunto de usuarios con comportamientos similares en cuanto al uso del Campus Virtual.

La función toma como parámetro tan solo el número de clústeres a generar, el cual será siempre mayor de cero, ya que se hace una validación previa a la ejecución de esta función. La implementación comienza de forma similar a la anterior, instanciando el **CSVLoader** y el fichero a procesar. Posteriormente se instancia el objeto **SimpleKMeans**, con el cual se establecen los parámetros del algoritmo, tales como el número de clústeres (a partir de la variable pasada como argumento), número de iteraciones, etc. A continuación, se debe crear un objeto **ClusterEvaluation** con nuestro objeto kmeans y el objeto **Instances**, y finalmente un objeto **ClustererAssignmentsPlotInstances**, que es el componente visual similar a PlotData2D en la función anterior. El resto del código es idéntico al anterior: se añade el componente visual a un **VisualizePanel**, éste a un JFrame que se muestra por pantalla, y finalmente se genera el fichero jpg.

La idea de crear un fichero jpg a partir de la gráfica mostrada puede ser útil para añadir una funcionalidad de historial de diagramas generados, pero su sentido tiene relación con una de las versiones anteriores de la aplicación, donde estas funcionalidades estaban implementadas de forma distinta, y se explicará en secciones posteriores en la memoria.

```
public void clustering(int numClusters) {
	CSVLoader loader = new CSVLoader();
	File file = new File("LogsAmpliado.csv");
	try {
		loader.setSource(file);
		Instances data = loader.getDataSet();

		SimpleKMeans kmeans = new SimpleKMeans();
		kmeans.setNumClusters(numClusters);
		kmeans.setMaxIterations(100);
		kmeans.setPreserveInstancesOrder(true);

		kmeans.buildClusterer(data);

		ClusterEvaluation eval = new ClusterEvaluation();
		eval.setClusterer(kmeans);
		eval.evaluateClusterer(data);

		ClustererAssignmentsPlotInstances plotInstances = new
	ClustererAssignmentsPlotInstances();
		plotInstances.setClusterer(kmeans);
		plotInstances.setInstances(data);
		plotInstances.setClusterEvaluation(eval);
		plotInstances.setUp();

		VisualizePanel vp = new VisualizePanel();
		vp.addPlot(plotInstances.getPlotData(kmeans.getClass()
	.getName()));

		JFrame jf = new JFrame("Diagrama");
		jf.setSize(1024, 512);
		jf.add(vp);
		jf.setLocationRelativeTo(null);
		jf.setVisible(true);

		// Guardar diagrama en fichero jpg
		File file1 = new File("src\\imagen.jpg");
		JPEGWriter writer = new JPEGWriter(vp, file1);
		writer.generateOutput();

	} catch (IOException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	} catch (Exception e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
}
```

7.3. Implementación de la función 'Visualize'.

8. Implementación de nuevas funcionalidades

A continuación, se detallan los cambios que se tendrán que realizar sobre la aplicación si se desea añadir una de las funcionalidades de Weka no añadidas.

En primer lugar, se debe añadir una función en la clase WekaDriver, que implemente el algoritmo mediante las clases necesarias de Weka. Se tendrá que añadir al controlador una función que instancie WekaDriver y llame al algoritmo que hemos escrito. Se deberá añadir un valor al input "algoritmo" de la interfaz para que el usuario seleccione esta nueva funcionalidad, y finalmente modificar la función actionPerformed para que, si se ha elegido esta opción, se llame a la función del controlador que hemos creado.

Otra característica que se puede ampliar, para poder realizar consultas más complejas, y visualizar resultados con más detalle, es el tema de parámetros de filtrado. Resultaría interesante poder filtrar por otros parámetros que no sean fecha y hora; visualizar el uso de un solo componente (por ejemplo, foros), o visualizar información solo relativa a entregas de proyectos, etc. Para implementar nuevos filtros debe modificarse el script de limpieza de datos, y añadirse condiciones en cada fila iterada, para decidir si se procesa o no en base al filtro introducido como argumento.

Finalmente, se puede mejorar la aplicación integrándola con otras herramientas que complementen los resultados con estadísticas y otra información adicional. Si ampliamos, además, el modelo de datos con más información, como resultados de exámenes, datos sobre ausencias en clase, etc., puede añadirse un componente de predictibilidad a la herramienta, de forma que esta pueda predecir estos datos adicionales en base al uso que se hace del Campus.

9. Diseño de la interfaz principal

Esta sección explica y muestra en detalle el diseño de la interfaz gráfica de la aplicación.

La interfaz está compuesta por una cabecera, una serie de inputs y un botón de 'obtener resultados'. A continuación, se muestran capturas de pantalla de la aplicación.

La figura 9.1 muestra la pantalla como se encuentra tras ejecutar la aplicación.

Figura 9.1. Interfaz principal.

Si el usuario selecciona 'Visualize' en el input de algoritmo se despliegan tres nuevos inputs. La interfaz resultante se encuentra en la figura 9.2. En la figura 9.3 pueden observarse los parámetros que se pueden seleccionar en cada input.

Si el usuario selecciona 'Clustering' aparece un solo input donde el usuario puede introducir el número de clústeres en los que se va a agrupar la información. El resultado de seleccionar dicho algoritmo se muestra en la figura 9.4.

Figura 9.2. Interfaz tras seleccionar algoritmo 'Visualize'.

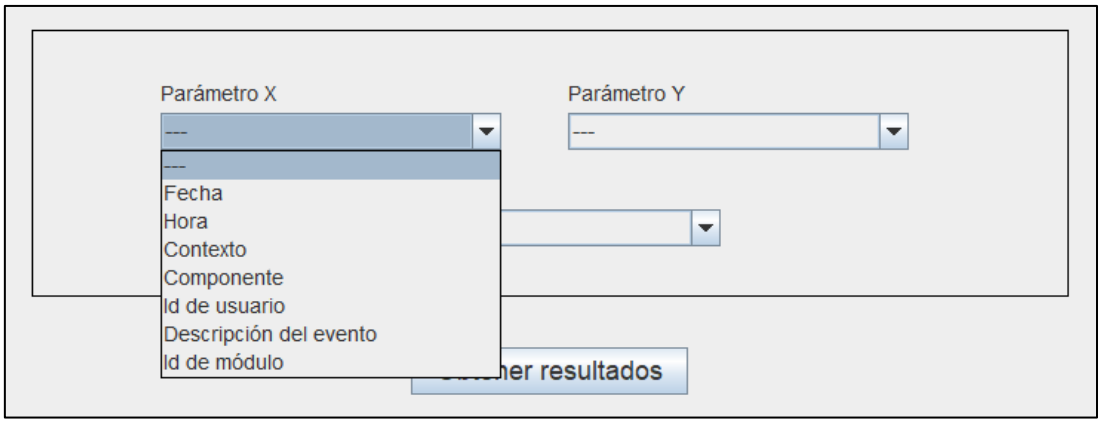


Figura 9.3. Despliegue de selección de Parámetro X.

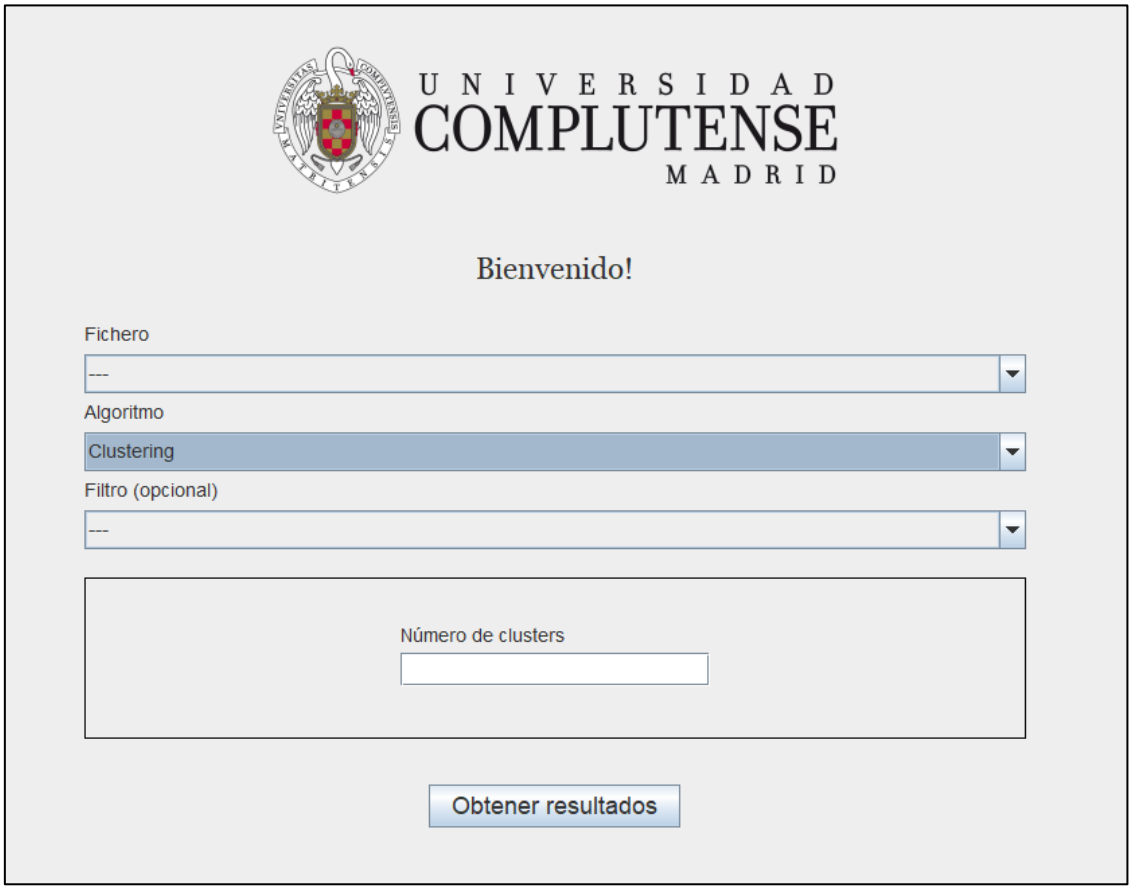
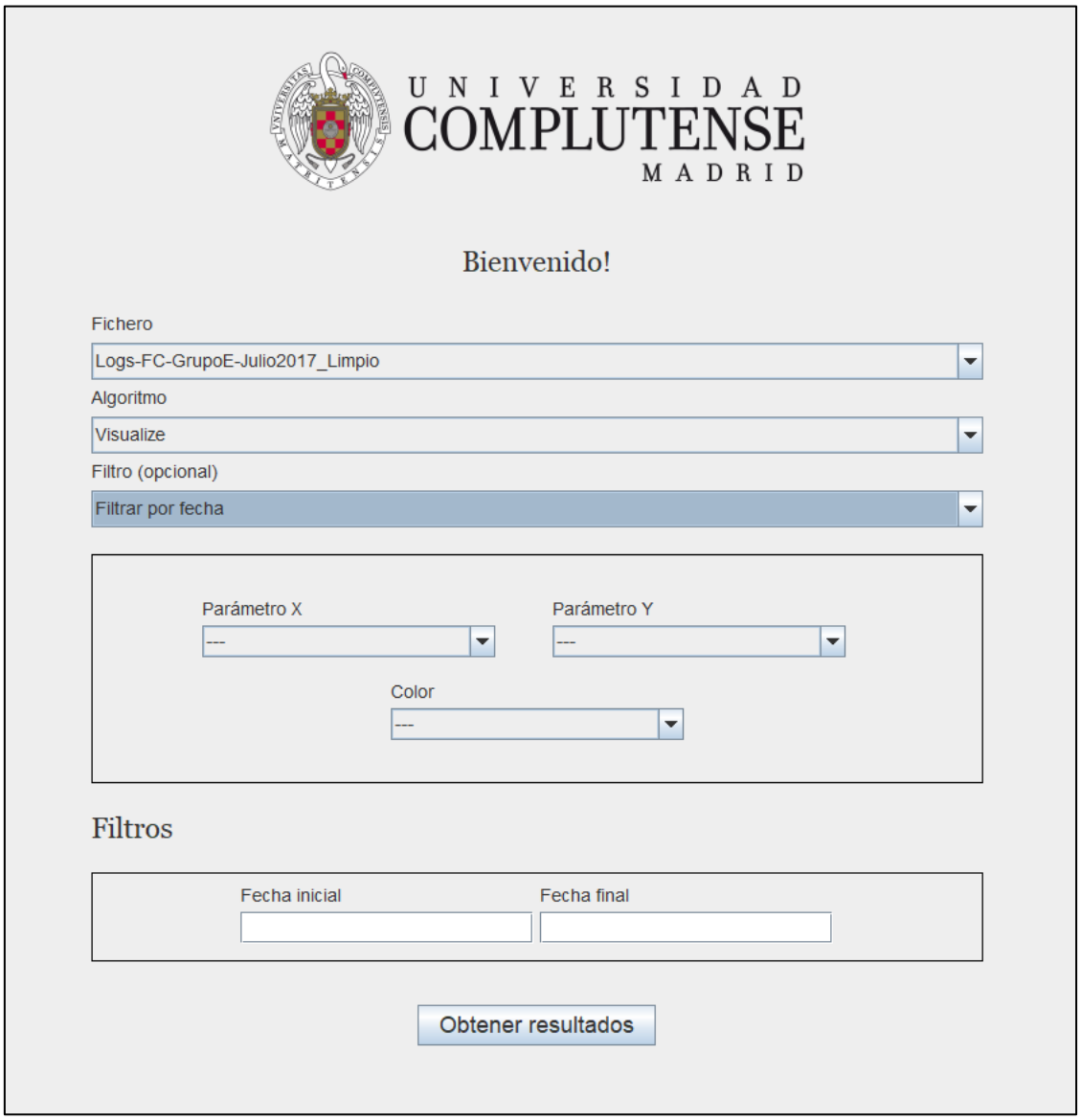


Figura 9.4. Interfaz tras seleccionar algoritmo 'Clustering'.

Al seleccionar un filtro, la interfaz despliega una nueva sección, que es igual para los dos filtros implementados, cambiando solo las etiquetas de los inputs. Esta sección tiene como título 'Filtros' y contiene una entrada para la fecha u hora inicial y una entrada para la fecha u hora final. En la figura 9.5 se muestra la interfaz con dicha sección integrada, junto con la sección de parámetros de 'Visualize'.

En la siguiente sección se explican distintos casos de prueba que se han llevado a cabo junto con capturas de los resultados obtenidos para cada uno.



9.5. Interfaz con todas las secciones integradas.

10. Visualización de los resultados obtenidos

Esta sección muestra los resultados de ejecutar principalmente dos tipos de pruebas: pruebas erróneas, es decir, donde falten datos por imputar o los datos sean incorrectos, y pruebas exitosas, que generen el diagrama correctamente.

La figura 10.1 muestra un mensaje que aparece como resultado de clicar en 'Obtener resultados sin haber rellenado los campos 'Fichero' y 'Algoritmo'. La aplicación mostrará el mismo mensaje si se deja cualquiera de estos en blanco.

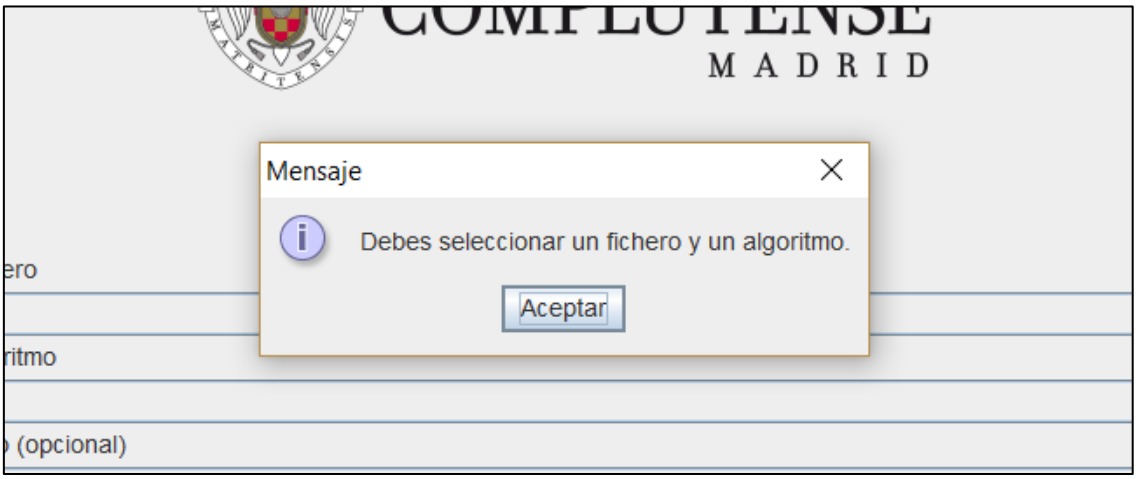


Figura 10.1. Captura de mensaje de validación de datos.

Este mensaje cambia en función de la información que falte por completar. Si el usuario selecciona el algoritmo 'Visualize' y no rellena los parámetros desplegados el mensaje será "Debes seleccionar un valor para cada parámetro". Si selecciona 'Clustering' y deja el campo vacío mostrará "Debe insertar un número de clústeres"; si rellena este campo, pero el valor es incorrecto (número menor o igual a cero, o dato no numérico) el mensaje mostrará "El campo 'número de clústeres' debe contener un número mayor que cero".

Cuando el usuario complete los datos de entrada de forma correcta, la aplicación mostrará el diagrama generado como se muestra en la figura 10.2.

Figura 10.2. Diagrama generado tras aplicar la función de 'Visualize'.

Esta gráfica tiene 'fecha' como parámetro *x*, 'contexto' como parámetro *y*, y componente como parámetro *color*. Una vez generado el diagrama, el panel permite cambiar los parámetros *x* e *y*. El panel inferior muestra a que valor corresponde cada color en el gráfico, y el panel de la derecha, al igual que el panel superior, permite cambiar los atributos de los ejes *x* e *y*.

Si se clica en algún punto, la aplicación muestra una ventana con el valor de los atributos en el punto seleccionado, tal como se muestra en la figura 10.3.

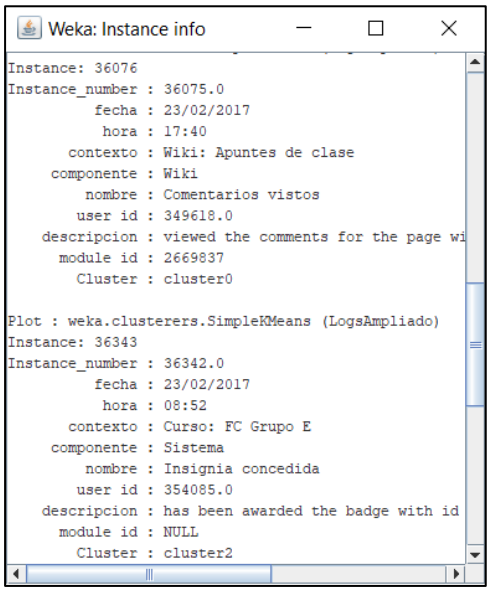


Figura 10.3. Información de una instancia

A continuación, en la figura 10.4 se muestra un ejemplo de 'clustering'. El diagrama tiene como parámetro *x* la descripción y como parámetro *y* el id del usuario. El color indica el clúster al que pertenece.

También se ha decidido mostrar una última imagen con las dos interfaces (la interfaz principal y los resultados obtenidos) integradas en una sola captura de pantalla. Se muestra en la figura 10.5.

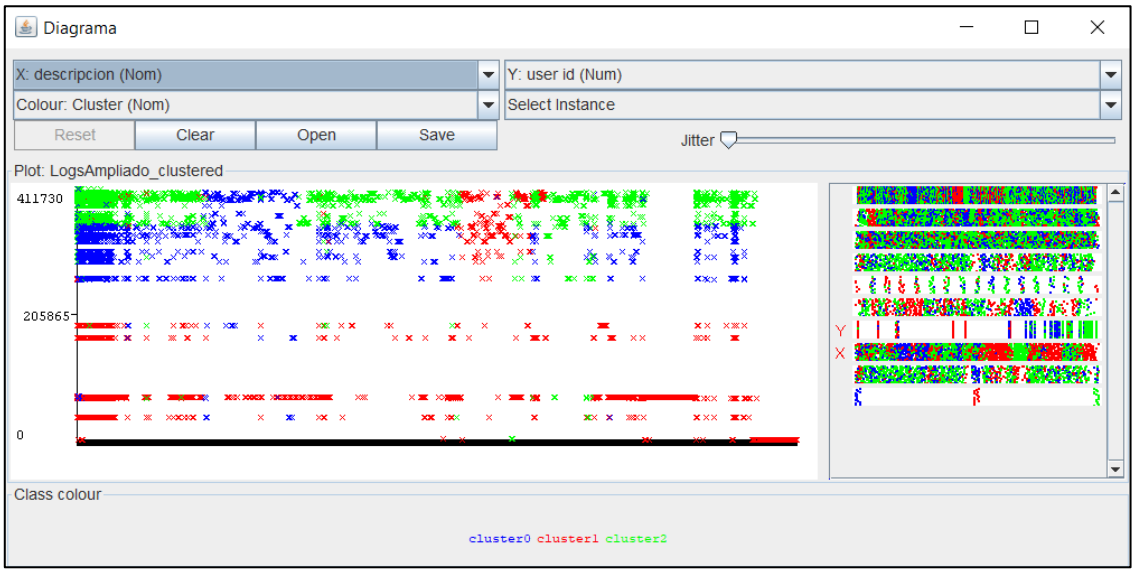


Figura 10.4. Diagrama generado tras aplicar la función de 'Clustering'.

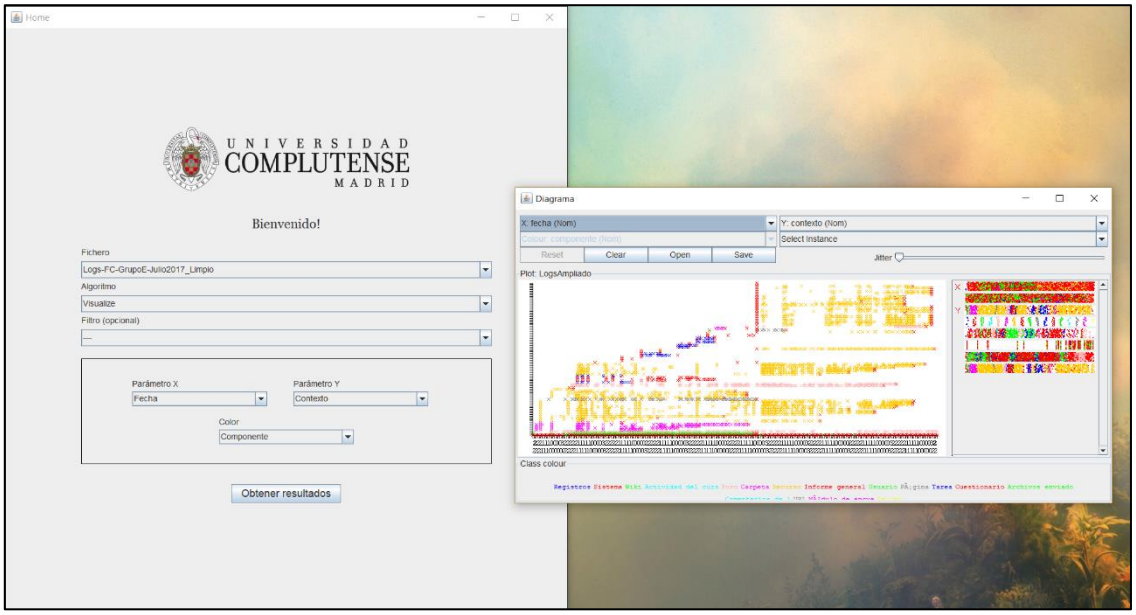


Figura 10.5. Captura con la interfaz de la aplicación completa.

11. Resultados experimentales

Teniendo en cuenta que para el desarrollo de la aplicación sólo se ha tenido un fichero de un curso como datos de entrada, las conclusiones que se puedan obtener a partir de los análisis realizados no son aplicables al uso general del Campus Virtual, o extrapolables a otras asignaturas, o años académicos.

Además, un análisis fiable de estos resultados debe ser realizado por el encargado de administrar un curso, ya que este tiene información adicional que es relevante a la hora de sacar conclusiones (por ejemplo, un diagrama puede parecer indicar que los alumnos han dejado de utilizar una herramienta a partir de una fecha, cuando la realidad es que esta herramienta puede haber sido deshabilitada por el administrador).

Teniendo estos dos factores en cuenta, a continuación, se exponen varias observaciones tras ejecutar distintas pruebas, variando los atributos de los ejes *X* e *Y* y el parámetro *Color*, así como los parámetros de filtrado.

En la figura 11.1 se muestra un diagrama que tiene utiliza *Fecha* como *Parámetro X*, *Hora* como *Parámetro Y*, y *Componente* como *Parámetro Color*. El diagrama comprende todo el año académico.

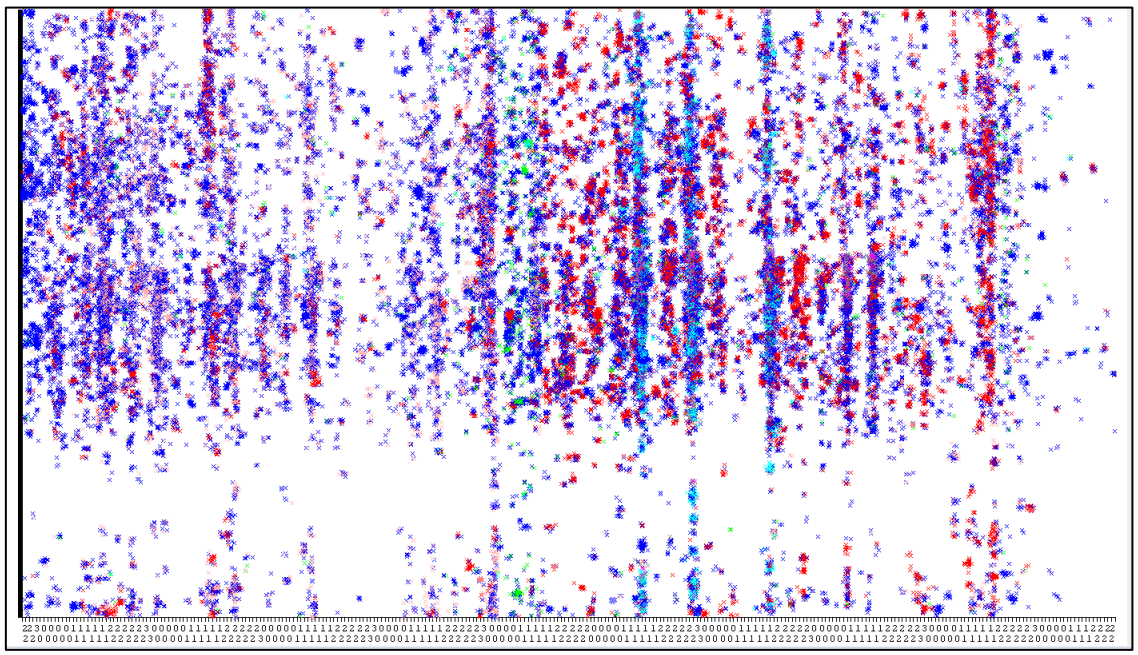


Figura 11.1. Resultados experimentales I

A simple vista, el diagrama no ofrece mucha información, debido a que, al haber tanta información, no se muestran los valores en los ejes *X* e *Y*. Sin embargo, si se clica en los puntos y si se conoce los valores que corresponden a cada color (aparecen en la interfaz como se muestra en la sección anterior, en la figura 11.1), se pueden hacer algunas observaciones.

- La franja horaria donde hay menos actividad corresponde, aproximadamente, al tiempo entre las 2 y las 8.30 de la mañana.
- Los puntos azules representan el acceso al curso. Por eso superan en tanta medida a cualquier otra actividad.
- Los alumnos solo hacen uso de la Wiki a partir del segundo cuatrimestre.
- Los puntos verdes representan la visualización de notas en el campus. Se puede observar que hay menos en los exámenes de junio que en los de febrero. Esto puede deberse a que se presenten menos alumnos o a que estos consulten las notas en la plataforma UCMnet.
- Parece haber menos actividad en periodos como Navidad, Semana Santa, y obviamente en verano.

A continuación, en el diagrama de la figura 11.2, se puede observar de forma más clara el uso de los distintos componentes del curso en el Campus a lo largo del año. El eje *X* representa la fecha, el eje *Y* el componente, y el *Color* representa el contexto. Una vez más, para poder obtener información detallada es necesario clicar en los puntos.

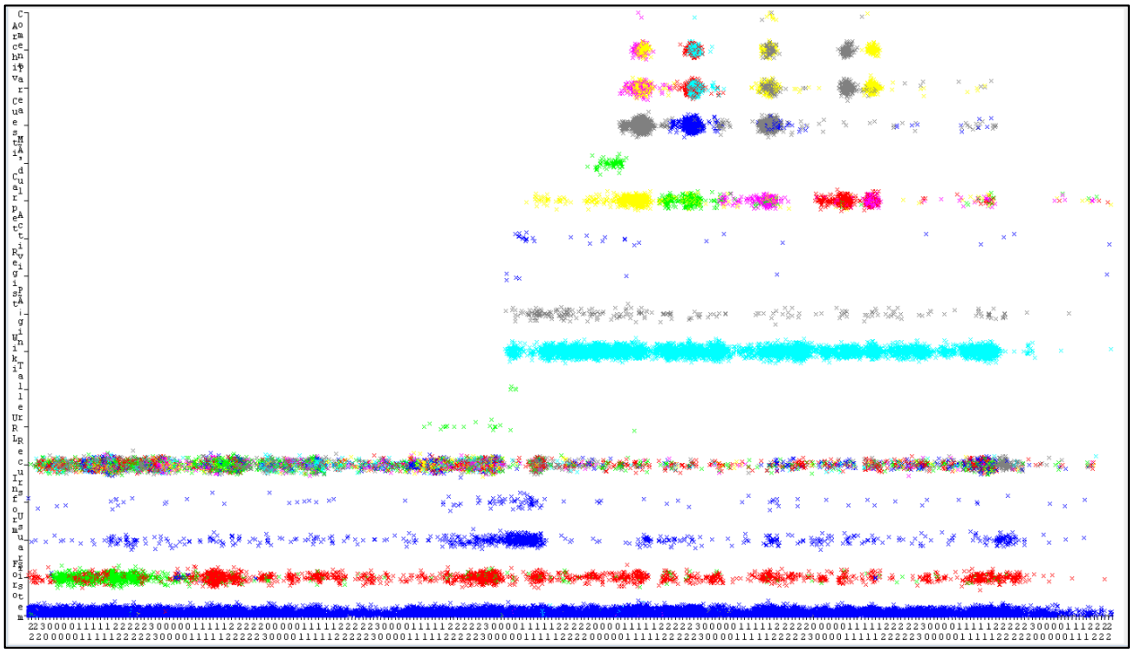


Figura 11.2. Resultados experimentales II.

La información que puede extraerse de esta gráfica es la siguiente:

- Se empieza a hacer uso de la Wiki a partir del segundo cuatrimestre, como se ha mencionado previamente.
- Se hace uso del foro durante todo el año, pero en menor medida en el segundo cuatrimestre.
- Durante el segundo cuatrimestre se realizan entregas de prácticas. Cada práctica tiene además una entrega adicional para las modificaciones, además de un cuestionario. Se observa que para la práctica 4 hay más tiempo entre entrega principal y entrega de modificaciones, y además esta no dispone de cuestionario. A simple vista se realizan más envíos en las prácticas 1 y 2 que en la 3 y 4.
- Se accede a la sección de "clasificaciones" durante todo el curso, pero en mayor medida después de los exámenes del primer parcial. No se hace un uso superior tras los exámenes de junio (probablemente los alumnos comprueben sus notas en UCMnet).

Estas observaciones son bastante generales, y su función es ilustrar algunos de los análisis que se pueden hacer de los resultados obtenidos. Como se ha explicado en la introducción de este documento, el objetivo del proyecto es meramente diseñar e implementar la herramienta. El análisis de estos datos debe hacerse por un docente que tenga más información sobre la asignatura y su curso correspondiente en el Campus.

12. Problemas encontrados durante el desarrollo

En esta sección se exploran los distintos obstáculos que han surgido durante el desarrollo de la aplicación, y como se han solucionado.

12.1. Problemas con la librería Weka

Uno de los pasos previos a empezar con el desarrollo fue encontrar una librería de algoritmos de Machine Learning, ya que el objetivo del trabajo era el análisis docente mediante aprendizaje automático, y no la programación de algoritmos complejos, por lo que era lógico utilizar una librería que ya tuviera los algoritmos necesarios.

La primera librería que se intentó utilizar fue TensorFlow, pero se descartó por su dificultad de integración con una aplicación de Java y su complejidad a la hora de programar. Weka pareció una opción válida para la implementación de algoritmos, por su interfaz gráfica que permite ver las funcionalidades y capacidades de la herramienta antes de empezar su programación e integración en Java. Sin embargo, ésta también supuso una serie de problemas, debido a que su fuente principal de documentación (la Weka Wiki) se mostraba inaccesible por mantenimiento. Por este motivo se tuvieron que realizar una gran cantidad de búsquedas de ejemplos de código que utilizasen Weka para entender bien el uso de sus librerías y funciones.

12.2. Formato de codificación de los datos de entrada.

A la hora de desarrollar el sistema de limpieza de datos de entrada, hubo un problema debido a que los ficheros de trazas no venían con la codificación correcta, y caracteres con tildes, así como la ‘ñ’ venían sustituidos por otros menos legibles. Esto hacía su lectura por parte de Weka imposible, por lo que se tuvieron que introducir mecanismos en el script para eliminarlos. Esto no sería un problema en un futuro si se utilizan ficheros que vengan codificados correctamente (UTF-8).

12.3. Implementación original de la interfaz

Otro problema que surgió fue debido a la idea original de desarrollar la herramienta como una aplicación web con Spring. Tras haber programado el script de procesamiento de

datos en Python, se creó una aplicación que utilizaba una interfaz desarrollada en HTML y CSS, muy similar a la interfaz actual. Sin embargo, como se ha podido observar en la sección de implementación, Weka trabaja integrando sus componentes visuales con Swing (Java). La solución que se pensó inicialmente fue transformar el componente visual generado en un fichero jpg y mostrarlo en la plantilla HTML, pero esto no dio resultado debido a que la clase JPEGWriter de Weka funciona únicamente para componentes visuales que están siendo mostrados por pantalla. Por tanto, se tuvo que descartar todo este planteamiento y desarrollar una nueva interfaz en Swing, lo cual supuso también un retraso en el desarrollo del proyecto.

Implementar la herramienta con Swing fue al final la opción acertada, puesto que los diagramas generados son dinámicos y pueden ser modificados al momento, y mostrar información adicional al interactuar con ellos. En la versión anterior, los diagramas eran estáticos y proporcionaban escasa información al usuario.

Otros problemas de menos importancia tuvieron que ver, sobre todo, con la falta de experiencia a la hora de programar en Python y Swing.

12. Conclusión

La conclusión que podemos obtener a partir de este trabajo es que con un refinamiento de la herramienta y aplicando las mejoras que se han ido indicando a lo largo de la memoria, el análisis de los datos de entrada con algoritmos de aprendizaje automático puede resultar viable y útil.

Mediante el uso de la aplicación, un profesor de una asignatura puede tener un vistazo rápido de la información relativa al uso del campus virtual por parte de sus alumnos, viendo así si estos están accediendo al material subido y si están haciendo uso de herramientas como foros, wikis, etc. De esta manera, puede ver si es necesario cambiar su enfoque a la hora de administrar la plataforma, y utilizarla de forma más inteligente, o intentar indicar a sus alumnos el uso que deben hacer de ella.

Los pasos principales seguidos para el desarrollo de este proyecto fueron: desarrollar un script para el procesamiento previo de datos, encontrar una librería capaz de analizar estos datos con algoritmos, diseñar una interfaz sencilla para el usuario e implementar las funcionalidades necesarias en una aplicación que integre todos los componentes anteriores. Los datos de entrada de la aplicación son ficheros de logs que indican acciones realizadas por alumnos en el campus virtual, tales como visualizar apuntes, acceder a foros y wikis, visitar perfiles de otros alumnos, etc. Se ha de recalcar también que, aunque la librería y algoritmos utilizados son de aprendizaje automático, la función de la aplicación es meramente analítica, y no de predictibilidad.

Las funcionalidades que se han implementado son la de visualizar los datos en un diagrama, y la de clasificar los datos en clústeres. Resultaría interesante ampliar la aplicación para obtener información adicional que pueda ser de utilidad para el profesor, como estadísticas, e implementar nuevos filtros para proporcionar información más detallada.

12. Conclusion

The conclusion that can be obtained from this work is that with a refinement of the software and applying the improvements that have been mentioned throughout the document, the analysis of the input data with machine learning algorithms can be viable and useful.

Using this application, a professor can take a quick view of the information relative to the use of the virtual campus by the students, seeing this way if these are accessing the materials that are being shared, and if they're making use of tools such as forums, wikis, etc. This way, teachers can see if they need to change their approach in terms of administrating the platform, and use it in a more intelligent way, or try to direct their students in the way they should be using it.

The main steps followed for the development of this project were: developing a script for the cleansing of data, finding a library capable of analyzing this data with algorithms, designing a simple interface for the user, and implementing the needed functionalities in an application that integrates all these components. The input data for the application are log files that indicate actions performed by students in the virtual campus, such as visualizing notes, accessing forums and wikis, visiting the profiles of other students, etc. It also must be highlighted that even though the techniques used are machine learning techniques, the function that the application serves is merely analytic, and not a function of predictability.

The functionalities that were implemented are visualizing the data in a diagram and classifying the data in clusters. It would be interesting to extend the application to feature additional information that can be of use by the professor, such as statistic information, or to implement new filters to provide more detailed information.

13. Bibliografía

Machine Learning:

1. What is Machine Learning? A definition - Expert System. (2019). Recuperado de https://www.expertsystem.com/machine-learning-definition/
2. Paloma Recuero, P. R. (s.f.-a). Los 2 tipos de aprendizaje en Machine Learning: supervisado y no supervisado. Fecha de consulta: 2019. Recuperado de https://data-speaks.luca-d3.com/2017/11/que-algoritmo-elegir-en-ml-aprendizaje.html
3. Deserción y permanencia en entornos MOOC. (2016, 30 de junio). Recuperado de http://www.mooc-maker.org/wp-content/files/WPD1.6_Informe_Final_ES_20_6_17.pdf
4. Analysis of Influencing Factors in Predicting Students Performance Using MLP - A Comparative Study (2015, 2 de febrero). Recuperado de https://pdfs.semanticscholar.org/bb03/fff265554fb512380156552e87cf22f45342.pdf
5. Google: Cloud AI. Fecha de consulta: 2019 desde https://cloud.google.com/products/machine-learning/
6. TensorFlow. Fecha de consulta: 2019 desde https://www.tensorflow.org/
7. TensorFlow. Install TensorFlow for Java. Fecha de consulta: 2019 desde https://www.tensorflow.org/install/lang_java

Documentación Weka:

8. Weka 3: Data Mining Software in Java. Fecha de consulta: 2018-2019 desde https://www.cs.waikato.ac.nz/ml/weka/
9. Weka (aprendizaje automático). (2017, 26 de diciembre). Wikipedia, La enciclopedia libre. Fecha de consulta: 2018-2019 desde https://es.wikipedia.org/wiki/Weka_(aprendizaje_autom%C3%A1tico)
10. York University. Weka Documentation. Fecha de consulta: 2018. Recuperado de https://www.eecs.yorku.ca/tdb/_doc.php/userg/sw/weka/doc/index.html

14. Anexos

1. Script de limpieza de datos, escrito en Python

```
# Librerías necesarias
import xlrd
import unicodecsv as csv
import sys
from datetime import datetime, timedelta

# Lectura del Excel con los logs
workbook = xlrd.open_workbook('./python/ficheros/' + sys.argv[1] +
'.xlsx')
worksheet = workbook.sheet_by_index(0)
num_rows = worksheet.nrows - 1
curr_row = 0

with open('LogsAmpliado.csv', 'wb') as csvfile:
   spamwriter = csv.writer(csvfile, delimiter=',', quotechar='"',
quoting=csv.QUOTE_ALL)
   spamwriter.writerow(["fecha", "hora", "contexto", "componente",
"nombre", "user id", "descripcion", "module id"])
   # Escritura en csv con las columnas ampliadas de la descripción
   while curr_row < num_rows:

      # Indice++
      curr_row += 1

      # Inicialización de los campos ya existentes
      fecha = worksheet.cell_value(rowx=curr_row, colx=0)
      hora = fecha.split(" ")[1]
      fecha = fecha.split(" ")[0]
      if (len(fecha) == 9):
         fecha = "0" + fecha
      n3 = worksheet.cell_value(rowx=curr_row, colx=1)
      if ('' in n3):
         n3 = n3.replace("", "") # Elimina caracteres especiales
      n4 = worksheet.cell_value(rowx=curr_row, colx=2)
      n5 = worksheet.cell_value(rowx=curr_row, colx=3)

      analizar = True

      # Filtrado de datos
```

```
      if (len(sys.argv) > 2):
         # Si se filtra por fecha
         if (sys.argv[2] == "fecha"):
            fechaInicio = datetime(int(sys.argv[3][6:]),
int(sys.argv[3][3:5]), int(sys.argv[3][0:2]))
            fechaFin = datetime(int(sys.argv[4][6:]),
int(sys.argv[4][3:5]), int(sys.argv[4][0:2]))
            if (fechaInicio > datetime(int(fecha[6:]), int(fecha[3:5]),
int(fecha[0:2])) or datetime(int(fecha[6:]), int(fecha[3:5]),
int(fecha[0:2])) > fechaFin):
               analizar = False
         # Si se filtra por hora
         if (sys.argv[2] == "hora"):
            now = datetime.now()
            horaInicio = datetime(now.year, now.month, now.day,
int(sys.argv[3][0:2]), int(sys.argv[3][3:])).time()
            horaFinal = datetime(now.year, now.month, now.day,
int(sys.argv[4][0:2]), int(sys.argv[4][3:])).time()
            if (horaInicio > datetime(now.year, now.month, now.day,
int(hora[0:2]), int(hora[3:])).time() or datetime(now.year, now.month,
now.day, int(hora[0:2]), int(hora[3:])).time() > horaFinal):
               analizar = False

      if (analizar):
         # Inicialización de los tres campos nuevos.
         a1 = ""
         a2 = ""
         a3 = "NULL"

         # Obtención del campo descripción e inserción de las palabras
en un array
         description = worksheet.cell_value(rowx=curr_row, colx=4)
         words = description.split(" ")

         # Obtención de campo "Código de usuario"
         a1 = int(words[4][1:-1])

         # Obtención de campo "Descripción del evento"
         i = 5
         while i < len(words) :
            if (words[i - 1] == "course") :
               break
            a2 += words[i] + " "
            i = i + 1
```

```
            a2 = a2.rstrip()

         # Obtención del campo "Código de módulo"
         if (i < len(words) and (words[i] == "module")):
            a3 = int(words[len(words) - 1][1:-2])

         # Generación de la fila con todos los campos, incluidos los ya
existentes
         row = [fecha, hora, n3, n4, n5, a1, a2, a3]

         # Escritura de todos los campos en csv
         spamwriter.writerow(row)
```